DOCTOR OF PHILOSOPHY

VLSI design and implementation of adaptive two-dimensional multilayer neural network architecture for image compression and decompression

Raj, P. Cyril Prasanna

Award date:
2010

Awarding institution:
Coventry University
M S Ramaiah University of Applied Sciences

VLSI DESIGN AND IMPLEMENTATION OF ADAPTIVE TWO-DIMENSIONAL MULTILAYER NEURAL NETWORK ARCHITECTURE FOR IMAGE COMPRESSION AND DECOMPRESSION

P. Cyril Prasanna Raj

A thesis submitted in partial fulfillment
of the University's requirements
for the Degree of Doctor of Philosophy

APRIL 2010

Coventry University

Research work carried out at
M. S. RAMAIAH SCHOOL OF ADVANCED STUDIES,
BANGALORE

COVENTRY UNIVERSITY

Doctoral Programme

Certificate

This is to certify that the PhD dissertation titled "VLSI Design and Implementation of Adaptive Two-Dimensional Multilayer Neural Network Architecture for Image Compression and Decompression" is a bonafied record of the work carried out by P. Cyril Prasanna Raj in partial fulfillment of requirements for the award of Degree of Doctor of Philosophy, Coventry University.

April – 2010

Declaration

VLSI Design and Implementation of Adaptive Two-Dimensional Multilayer Neural Network Architecture for Image Compression and Decompression

The thesis dissertation is submitted in partial fulfillment of academic requirements for Doctor of Philosophy Degree of Coventry University in Electronics and Computer Engineering. This dissertation is a result of my own investigation. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature **:**

Name of the Student **:** **P. Cyril Prasanna Raj**

Date **:** **April, 2010**

Acknowledgement

The successful completion of any task will be incomplete without complementing those who made it possible and whose guidance and encouragement made my efforts successful. The right way to express my gratitude is this section of my research work. I take immense pleasure and am overwhelmed at this point of time to express my sincere and heartfelt gratitude to all who were involved with me in successful completion of my research work.

With deep sense of gratitude I acknowledge the help and encouragement rendered by my President, **Dr. S. R. Shankapal**, Director, MSRSAS, Bangalore, for his valuable suggestions, support, encouragement and guidance. His advice has been my inspiration throughout my work at MSRSAS. I am grateful for his valuable feedback and suggestions throughout my career at MSRSAS and also successful completion of this work. I am and will always be indebted to him.

This research work has benefited from the help of many individuals. The discussion sessions and guidance of **Dr. S. L. Pinjare,** was key in realizing and improving this research work. I would like to thank him for his advice, guidance and support during the course of research. His timely support and motivation has been one of the most important aspects towards successful completion of this research work.

I am thankful to **Dr. Y. A. Vershinin**, Professor, Coventry University, U.K, for accepting to be my CU supervisor, his thought provoking suggestions and also key inputs provided during my thesis writing have helped me in successfully completing this work.

I would like to thank **Dr. M. D. Deshpande,** Head Research**,** MSRSAS, and Dr. **Govind R. Kadambi**, Dean (Academics), MSRSAS, Bangalore for providing necessary resources, facilities and excellent environment necessary for successful completion of this research. Their politeness in conduction of research reviews has really benefited me to complete the research work successfully.

I take this opportunity to express my gratitude to **Dr. Peter White**, Professor - Coventry University. During the review meetings his suggestions and directions have

helped me a lot in understanding the purpose of research. I am ever grateful to him for his kindness and patience listening during review meetings.

I also take great pleasure in expressing my thanks to **Dr. A. G. Ananth**, Respond Office, ISRO, Bangalore, **Shri. E. Devadoss**, Deputy Director, Respond Office, VSSC, and **Prof. Krishna Prasad** and his team, Bangalore for their support throughout the research work.

I take this opportunity to thank my colleagues at **EEE department faculty members and students** who have assisted me in carrying out this research and helped me in validating the results.

I would like to thank the following distinguished members who have constantly supported and helped me in completing the research work:

1. Ms. Sarojini, Head DSD, VSSC, Trivandrum
2. Shri. B. G. Suresh, Director, VSSC Trivandrum
3. Dr. T. V. Ananthapadmanabhan, MSRSAS

I wish to express my warm and sincere thanks to all my teaching staff, non-teaching staff at MSRSAS who directly or indirectly have been of great help for the successful completion of the thesis.

I would also thank my parents (my mother Mrs. G. Shanthy) for their unconditional love and support without which nothing would have been possible.

I take this opportunity to express my deepest and unconditional love to my wife Sujatha Cyril and my daughters Advika C. Sana and Arshia C. Sana without their support my research work would have not been able to see the light. I am always indebted to them and very sincerely acknowledge their patience exhibited during this period. Their support has motivated me in successful completion of my research work.

Above all I thank the almighty, for giving me strength and an opportunity to pursue my research work. It is his blessings that have guided and encouraged me all through my difficulties in successfully completing this work.

Abstract

In this research, adaptive Two-Dimensional Multilayer Neural Network (TDMNN) architecture is proposed, designed and implemented for image compression and decompression. The adaptive TDMNN architecture performs image compression and decompression by automatically choosing one of the three (linear, nonlinear and hybrid) TDMNN architectures based on input image entropy and required compression ratio. The architecture is two-dimensional, 2D to 1D reordering of input image is avoided, as the TDMNN architecture is implemented using hybrid neural network, analog to digital conversion of image input is eliminated. The architecture is trained to reconstruct images in the presence of noise as well as channel errors.

Software reference model for Adaptive TDMNN architecture is designed and modeled using Matlab. Modified backpropagation algorithm that can train two-dimensional network is proposed and is used to train the TDMNN architecture. Performance metrics such as Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) are computed and compared with well established DWT-SPIHT technique. There is 10% to 25% improvement in reconstructed image quality measured in terms of MSE and PSNR compared to DWT-SPIHT technique. Software reference model results show that the compression and decompression time for TDMNN architecture is less than 25 ms for image of size 256 x 256, which is 60 times faster than DWT-SPIHT technique.

Based on weights and biases of the network obtained from the software reference model VLSI implementation of adaptive TDMNN architecture is carried out. A new hybrid multiplying DAC is designed that multiplies current intensities (analog input) with digital weights. The hybrid multiplier is integrated with adder and network function to realize a hybrid neuron cell. The hybrid neuron cell designed using 1420 transistors works at 200 MHz, consuming less than 232 mW of power, with full scale current of 65.535 µA. Multiple hybrid neurons are integrated together to realize the 2-D adaptive multilayer neural network architecture.

Contents

List of Figures

List of Tables

Nomenclature

C	Capacitance, F
I_{bias}	Bias current, A
I_{ds}	Drain current MOSFET (Ampere), A
I_s	Static Current (Ampere), A
L	Inductance, H
$L_{channel}$	Channel length, m
P	Power gain in dB, output power, dBm
Q	Quality Factor
R	Resistance, Ω
V_{bias}	Bias voltage, V
V_{gs}	Gate-to-source voltage, V
V_t	Threshold voltage, V
W	Channel width, m
X	Reactance, Ω
Z	Impedance, Ω
α	Closed Loop Gain
ω	Angular frequency, rad/s
φ_n	Phase noise, dBc/Hz
f	Frequency, Hz
g_m	Transconductance, Ω^{-1}
r_{ds}	Drain-to-source resistance, Ω
μA	microampere
μs	micro (10^{-6}) sec
ns	nano (10^{-9}) sec
mm	millimeter
MHz	Mega Hertz (Frequency)
mW	milli (10^{-3}) Watt
W	Watts
p_i, v_i	Input to NN (Volt)
t_i	Target input (Volt)
v_g	Gate voltage with respect to bulk (Volt), V
v_s	Source voltage with respect to bulk (Volt), V

Abbreviations

1-D	One-dimensional
2-D	Two-Dimensional
ADC	Analog to Digital Converter
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASIC	Application Specific Integrated Circuit
BP	Backpropagation
BPNN	Backpropagation Neural Network
CCD	Charge Coupled Device
CMOS	Complementary MOS
CR	Compression Ratio
DAC	Digital to Analog Converter
DWT	Discrete Wavelet Transform
DRC	Design Rule Check
EDA	Electronic Design Automation
EP	Error Propagation
FFN	Feed Forward Network
FPGA	Field Programmable Gate Array
GB	Giga Byte
GDSII	Graphical Data Source Interchange
IDWT	Inverse Discrete Wavelet Transform
JPEG	Joint Pictures Experts Group
LAN	Local Area Network
LVQ	Learning Vector Quantization
LVS	Layout versus Schematic
MATLAB	Matrix Lab
MaxError	Maximum Error
Mbps	Mega bits per second
ML	Multilayer
MLP	Multilayer Perceptron
MPEG	Motion Pictures Experts Group
MSE	Mean Square Error
MOS	Metal Oxide Semiconductor
MOSFET	MOS Field Effect Transistor
MVBP	Modified Vogl Back Propagation
NA	Neural Architecture
NN	Neural Network
Op	Output of NN
PCA	Principal Component Analysis

PSNR	Peak Signal to Noise Ratio
RRANN	Reconfigurable Neural Network
SNR	Signal to Noise Ratio
SPIHT	Set Partitioned Integer Hierarchical Tree
SOM	Self Organizing Map
T	Target for Learning
TDMNN	Two-Dimensional Multilayer Neural Network
TSMC	Taiwan Semiconductor Manufacturing Company
TV	Television
VBP	Vogl Back Propagation
VLSI	Very Large Scale Integration
VSSC	Vikram Sarabhai Space Centre
W/L	Width / Length

Chapter 1 - Introduction

Today's technological development has intruded into human lifestyle to such an extent that we feel uncomfortable without the use of electronic gadgets, information systems, household electronic equipments and display systems. We have started adopting and have got used to the latest electronic gadgets without which we feel isolated from rest of the world. The main reason to this is that, new products developed are sophisticated and addresses the needs of common man and these products are user-friendly and reliable. The electronic equipment available interacts with us and assists us in our day to day activities. Most of the commercially available electronics equipments have graphical display interfaces. Data in the form of images, motion pictures, icons and text displays on graphical displays assist humans and hence language or speech does not pose a barrier for the use of these gadgets worldwide. Visual representation tends to be perceived as being more efficient than the spoken or written word. As the demand for user-friendly equipment keeps increasing, image processing finds utmost importance. New applications based on image processing are being invented. Image processing of biomedical signal assist doctors and technicians to diagnose and treat patients, similarly processing of satellite images assist scientists to monitor and predict climate changes. In a time critical application such as launching of satellites, image processing assists in monitoring the launch of satellite using the launch vehicle from the base station. Controlling of launch vehicle is monitored based on image signal received at the base station. One of the major challenges working with image data is the size of image, or the number of bits required to represent visual information. A typical color image of size 512 x 512 consists of 6.5 Mega bits. Real time processing of image data, storage and transmission of image signals always pose a challenge for designers. Image enhancement, image restoration, image segmentation and image compression are the major image processing operations. In this work, compression of images based on neural network architectures for launch vehicle applications are addressed.

1.1 Preamble

Images that we see with our visual system are captured using image sensors in digital form and stored in memory banks. As the storage space for these digital samples consumes large space, cost increases. For example, a natural image captured from a sensor when digitized is converted to a 2-D matrix of size 256 x 256 pixels, each pixel representing intensity level of the natural image. Image sensors capture pixel intensities. Each pixel can be represented by 1 bit, 8 bit or 24 bit for black and white, gray scale and color image respectively. A color image of size 256 x 256 represented using 24 bit requires a storage space of 1.5 mega bits (256*256*24 = 1.5 Mb). A motion picture captured at 30 frames per second requires a storage space of 45 Mbps (1.5 Mb*30 = 45 Mbps). The storage space for a three hour movie requires 486 Giga bits (4.5 Mb*60*60*3 = 486 Giga bits). Table 1.1 present's bandwidth, transmission delay of three major communication links and transmission delay of uncompressed video signal.

Table 1.1 Data rate of communication channels

Communication Channel	Data Rate	Round Trip Time (RTT) or Transmission Delay	Transmission delay for uncompressed video = Data Size (486 Gb)/Data Rate
DSL/ADSL (William Stallings 2010)	256 Kbps to 40 Mbps	50 ms	202 minutes
Mobile broad band - HSDPA (Martin Sauter 2006)	1.2 Mbps to 84 Mbps	100 ms	96 minutes
High-speed terrestrial network (Kai Chen et al. 2003)	1 Gbps	1 ms	8 minutes

Compression of raw video data thus makes it possible for transmission of data through existing communication links with minimum delay. Compressed images are transmitted through communication channels and are decompressed on the receiver side, without affecting the quality of picture. However, an image or video data being transmitted in real time have a minimum amount of delay of few milli seconds (Stuart

Cheshire 1996). This delay is mainly due to three factors. Firstly, huge data need to be compressed to the available channel bandwidth. This compression is accomplished using sophisticated software's and hardware's executing complex algorithms on high speed architectures, and this constitutes the maximum delay (Schaphorst and Richard 1999). Secondly, the compressed data is further encoded with security bits and parity bits for security and to avoid noise as they travel through the channel. In order to ensure that the data reaches the destination, they are also padded with specific bits to guide the encoded information through the channel and reach the destination. This processing of the compressed data using software and hardware devices introduces delay (Stuart Cheshire 1996). Thirdly, the transmitted data traveling through the channel also introduces delay and is called as the channel delay (Stuart Cheshire 1996). Delay in a communication system of few micro seconds to few milliseconds (Umashankar 2003) is negligible and usually not observed and is accepted. On the other hand, in crucial applications such as satellite launching where necessary control actions needs to taken based on visual information, a small amount of delay causes a major impact on the system and decision making. For example, if the trajectory path of the launch vehicle carrying a satellite is being monitored using visual information being captured and down linked in real time by onboard video processing systems, guiding the launch vehicle and launching of the satellite to appropriate destination requires critical and time bound control system. Based on the received visual information at the base station if suitable control actions have to take place, the delay in decompressing the compressed data has its impact, as the entire application is time critical. In this case the delay in signal reception has a major impact. Channel noise affecting the decompressed image is another major challenge. In a personal interview conducted by the author, Ms. Sarojini, Head, Digital Signal Design, Avionics Group, VSSC, Trivandrum, mentioned about the draw backs of the system that they had which was used to monitor the launch vehicle. In the discussion she emphasized on the need for a system that can work at 40 frames per second and also is immune to channel noise (Sarojini 2004). Information in an image when compressed, any error due

to noise in the channel affecting the compressed bits significantly affects the decompressed images. This research work carried out is an approach to firstly, minimize the system delay in compression and decompression of image applicable to time critical applications with novel architectures being proposed, designed, implemented and validated for image compression. Secondly, with the compressed data traveling through noisy channel, noise interference on the compressed data impacts the decompressed output, hence it is required to improve image quality at the receiver even in the presence of channel noise or error.

1.2 Need for Multidimensional Neural Network Architecture

Many practical solutions have been designed by eminent scientists and researchers adopting conventional techniques for compression and decompression of still and motion pictures. These systems are interoperable worldwide as they support uniform standards such as JPEG and MPEG (Chrysafis and Ortega 1998, Frescura, F., Giorni, M., Feci, C. and Cacopardi, S. 2003, Lian *et al.* 2003 and Mitchell and Pennebaker 1993, Signoroni, Lazzaroni and Leonardi 2003). JPEG and MPEG standards recommend guidelines for compressing and encoding the image signals, frame formats for transmission of compressed data and hardware requirements. These standards assume that the input image is digitized raw data represented in bmp, png or tiff formats (William and Joan 2004, *JPEG 2000 image coding system* 2000).

Natural images are captured using either Charged Coupled Devices (CCD) sensors (Tompsett, M. F. Amelio, G. F. Bertram, W. J., Jr. Buckley, R. R. McNamara, W. J. Mikkelsen, J. C. and Jr. Sealer, D.A. 1971) or Complementary Metal Oxide Semiconductor (CMOS) sensors (D. Renshaw, P. B. Denyer, G. Wang, and M. Lu 1990). Light intensity that is reflected from the object(s) is focused through the lens and strikes array of image sensors as shown in Fig. 1.1. Photo detectors that form the sensor array capture the light intensities and convert into voltage levels. Both type of sensors capture light intensities and convert into electrical signals. Interfacing circuit reads the electrical

signals from sensor arrays and processes the captured light intensities. Interfacing circuits for CCDs and CMOS sensor have different configuration (Dave Litwiller 2001). In a CCD sensor, light intensity reflected from the object(s) is focused through the lens that energizes the CCD that is arranged in 2-D array (Spatial). CCD sensors capture the light intensity of the image being focused. The image data captured is read out of the CCD array in the form of voltage or current equivalents.

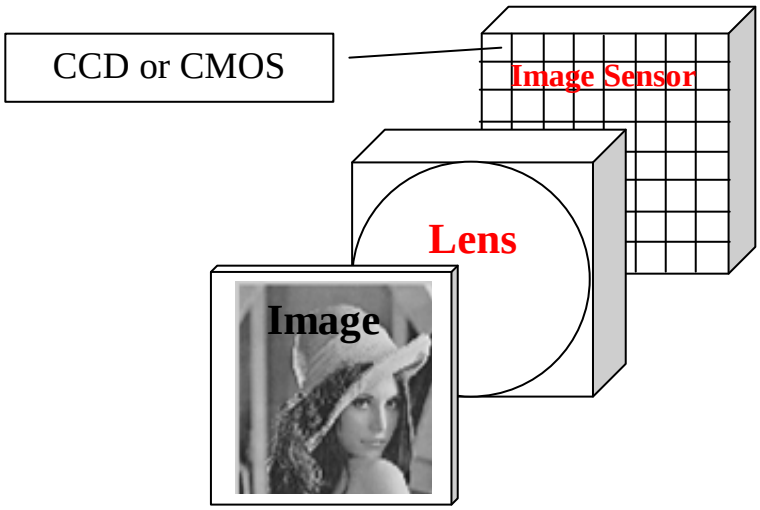


Figure 1.1 Image sensor

Fig. 1.2 demonstrates the read out operation of these intensities stored on the CCD devices (Karim Nice, Tracy V. Wilson and Gerald Gurevich 2004). Energized CCD array elements that accumulate charge corresponding to light intensity is read out serially and amplified by a gain factor (e.g. 10 μA/e). This way of serially reading out the data from the array reduces the complexity of drawing multiple wires from the array elements and also ensures the serialization of the data samples. The analog intensities read out from the matrix are further digitized to equivalent binary values in the Analog to Digital Converter (ADC) and stored in memory.

In a CMOS imaging sensor, arrays of active pixel sensors (Dickinson Alexander G., Eid El-Sayedi and Inglis Davida 1997) are placed at every pixel location that consists of a photo diode and active amplifier. Photo sensor captures light intensity and converts to voltage levels, which is converted to digital samples using ADC as shown in Fig. 1.3.

CMOS sensors have advantages compared to CCD sensors in terms of speed, responsivity, windowing, anti-blooming and reliability (Dave Litwiller 2001).

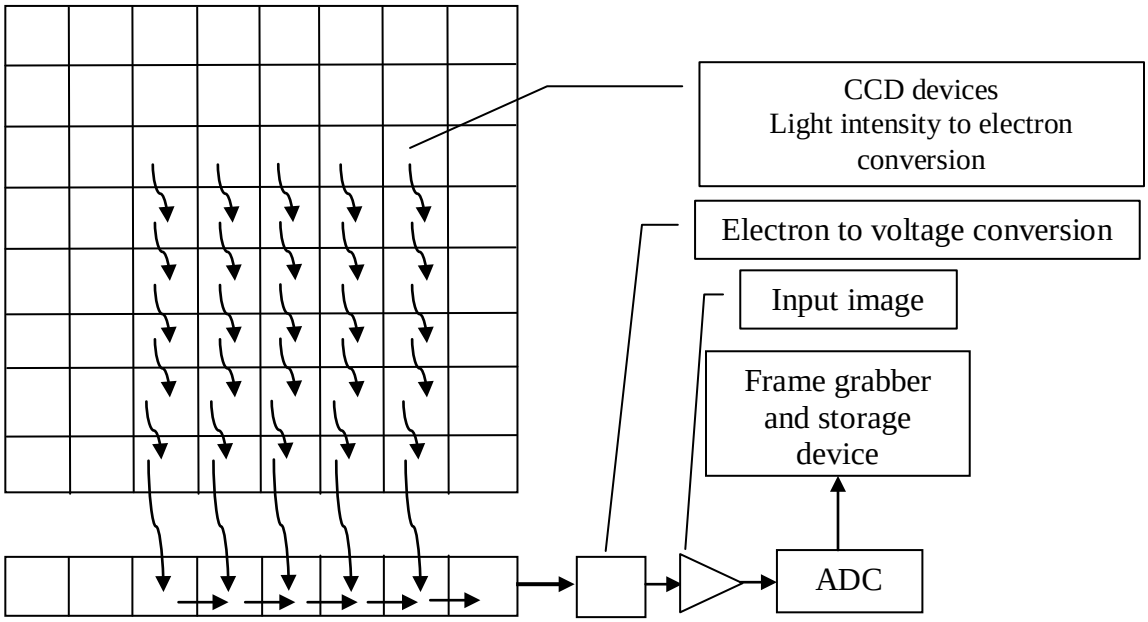


Figure 1.2 CCD based image sensor with analog readout

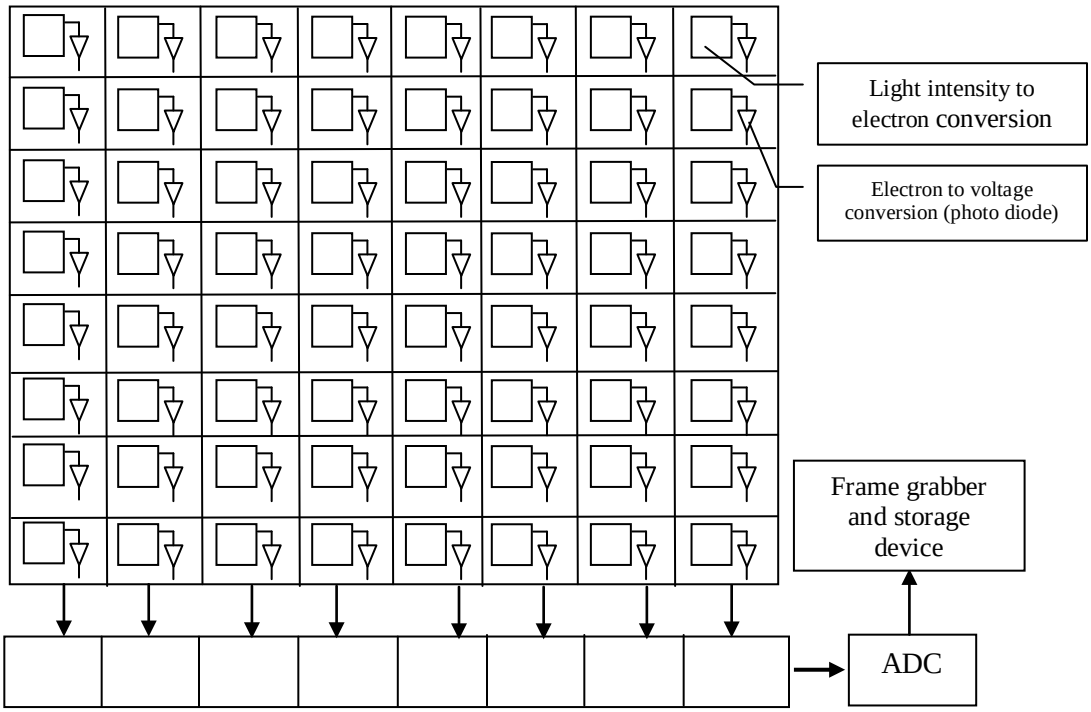


Figure 1.3 CMOS sensor based imaging unit

CCDs are superior to CMOS in terms of image quality and flexibility (Dave Litwiller 2001). Both technologies for imaging offer unique advantage, one common limitation is that, the voltage samples are read out serially to avoid number of wires from each sensor and additional electronic circuitry is required to convert analog to digital samples. Digital data read out of imaging unit which is in uncompressed form is stored in a digital memory.

Digital samples are in uncompressed form and occupy more space. Compression of these samples is performed using the classical techniques as per the guidelines recommended by JPEG or MPEG. Two observations are made at this point, which leads to the scope of this research work. First, the time involved in reading out the 2-D array elements into 1-D elements can be avoided if a 2-D architecture that can accept the 2-D samples and can process the 2-D signals and perform compression. Second, instead of processing the 1-D data in digital domain, what if we process the 2-D analog data directly read from CCD devices or CMOS sensors using 2-D architectures. In a CMOS sensor array, the output of photo diodes which is current (I) can be processed using the 2-D architecture proposed in this work. In case of CCD sensor array, it is required to add additional circuits at every pixel location to convert charge to voltage samples, these voltage samples can be directly interfaced to the 2-D architecture. These two observations are the motivating factors to carry out this work.

The 2-D architecture shown in Fig. 1.4 consists of image sensor array capturing brightness or intensity levels of image. These intensities are amplified and converted to equivalent current or voltage values. These N x N arrays of electrical samples are directly fed into a two dimensional architecture that processes the analog values and reduce the array dimension to M x M array elements. Since M < N, the number of pixel elements representing the image is compressed from N^2 to M^2 elements, which can be further encoded using 2-D techniques for storage and transmission. The advantage of 2-D architecture is that as the analog samples or pixel intensities are captured in the analog domain and processed using 2-D architecture, this avoids use of Analog to Digital

Converter (ADC) at the input side hence reduction in hardware complexity and also further reduces the delay.

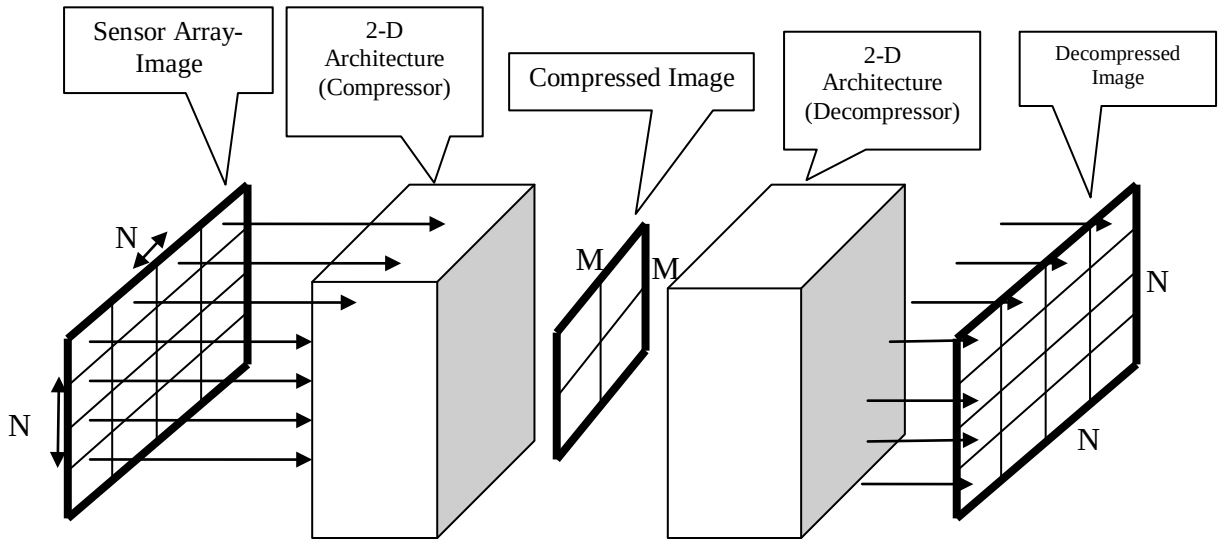


Figure 1.4 2-D architecture for image capture and process

1.3 Need for Neural Networks for Image Compression

Classical techniques for image compression based on the recommendations from JPEG standards have been implemented on hardware and software, and these techniques have been used in electronic products and applications (Andra, Chakrabati and Acharya 2003, Chiang, J. S., Lin, Y. S and Hsieh, C. Y. 2002, Fang *et al.* 2003, Iain 2002 and Ong *et al*. 2002). Fig. 1.5 shows the basic block diagram of image compression and decompression unit.

The major blocks for image compression are transform coding, quantization and entropy coding (David and Michael 2001). Input to the system is image data with N_1 bits per pixel, encoded data is the compressed out with N_2 bits per pixel. As N_2 is less than N_1, only N_2 bits are required to represent N_1 bits achieving compression. The compressed data N_2 that consists of the entire information of N_1 packed using N_2 bits is transmitted through the channel. On the receiver side, N_2 bits received is decompressed to N_1 bits. However, during the transmission, if the compressed data gets corrupted due to noise in

This image has been removed

Figure 1.5 Block diagram of image compression unit (David and Michael 2001)

Fig. 1.6 shows the block diagram of JPEG 2000 standard based image compression and decompression unit, the transmitter is on the launch vehicle and the receiver is at the base station. It is required to capture the image sequences on board and compress the captured data and transmit to the base station, at the base station, the image is decompressed and used to track launch vehicle movement and satellite launch activity from the base station.

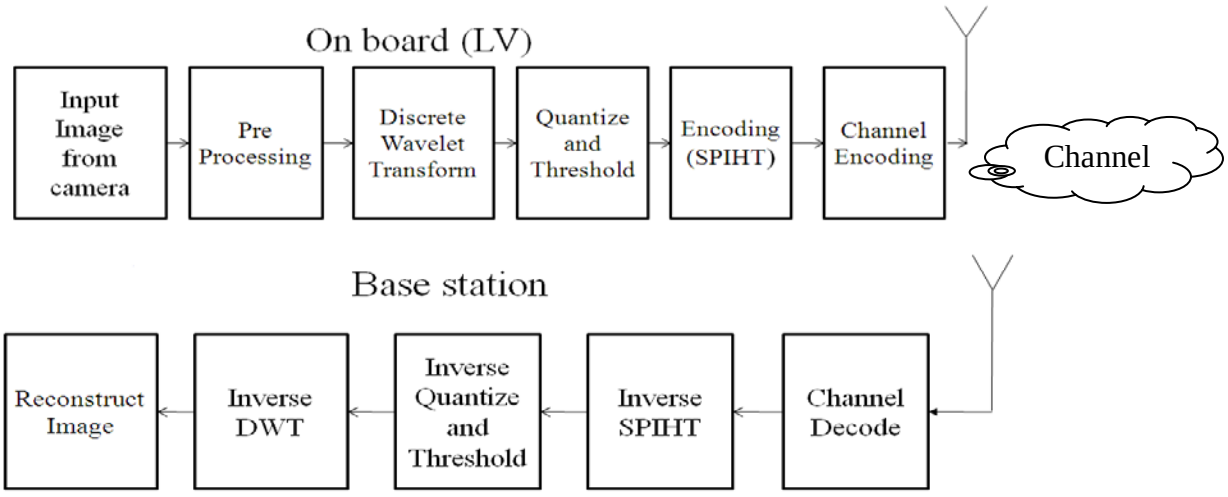


Figure 1.6 JPEG based image compression and decompression unit for launch vehicle applications

Launch Vehicle (LV) travels at a speed of 2 – 11 km/sec, launching the satellite into the orbit takes 10 minutes to 30 minutes (Chang IS 2007), thus it is required to monitor the satellite launch and onboard monitoring of launch vehicle trajectory. Camera mounted on the launch vehicle captures images, compresses and transmits to the base station. Based on the received visual information it is requried to take corrective feedback from the base station. Due to noise in the channel if compressed data is corrupted, reconstructed image at the base station is distorted. Fig. 1.10 to Fig. 1.14 demonstrates the results for image compression using Discrete Wavelet Transform (DWT) and Set Partitioned Integer Hierarchical Tree (SPIHT) encoding techniques. This work was carried out as part of research work sponsored by Vikram Sarabhai Space Centre (VSSC), Trivandrum, India. The aim of this work was to identify the impact of bit errors due to channel noise on the reconstruction of compressed image. Noise in the channel causes a bit '1' to become '0' or vice versa. If the compressed bit information is corrupted due to channel noise, decompressing the image from the corrupted packets of received information affects the quality of the image.

Figure 1.7 Input image (pictorial representation)

Image shown in Fig. 1.7 is of size 512 x 512 is compressed using DWT and SPIHT technique and is used as a test case to illustrate the impact of channel noise on reconstructed image. Fig. 1.8 shows the pixel representation of the input image shown in Fig. 1.7 (only a part of the 512 x 512 image is presented in Fig. 1.8). Every pixel is

represented using 8 bits or expressed as 8 bits per pixel (bpp) or in general N_1 bits per pixel.

110	82	105	83	93	83	110
123	93	83	110	80	93	93
146	119	105	80	105	71	83
142	83	119	86	59	93	74
146	119	119	93	83	59	71
137	93	105	83	119	80	89
110	82	83	82	54	93	71
119	127	119	77	93	54	93
93	119	93	83	96	93	83
119	137	110	119	80	93	69
93	119	146	105	69	83	105
105	110	93	110	105	59	119
110	123	137	142	83	105	93

Figure 1.8 Input image (pixel representation) represented using N_1 bpp

Image of size 512 x 512 represented using 8 bpp (N_1 bpp) consists of 512*512*8 (M_1) bits is represented using 512*512*0.1 (M_2) bits after compression. Compression of 0.1 bpp (N_2 bits per pixel) implies that every 10 pixels of the input image are represented using 1 bpp. The compressed image is binary stream of data (1010101010101010101101011010101……) that represents 512 x 512 images in compressed form with N_2 bpp. The compressed M_2 bits that contain the information of M_1 bits, when transmitted may get corrupted in the channel due to channel noise. Fig. 1.9 to Fig. 1.13 presents the reconstructed images with channel noise affecting the compressed data. In this work, channel noise is introduced by complementing the binary values at randomly chosen bit positions in the compressed binary stream (hence forth called as bit errors).

Figure 1.9 Reconstructed image after error being introduced at 1st and 100th bit positions [Max. Error = 167, MSE = 219.13, PSNR = 24.72]

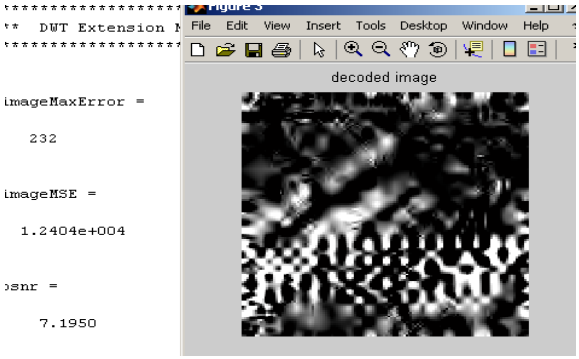


Figure 1.10 Reconstructed image after error being introduced at 10th and 30th bit positions [Max. Error = 232, MSE = 12404, PSNR = 7.19]

Figure 1.11 Reconstructed image after error being introduced at 10th and 50th bit positions [Max. Error = 187, MSE = 230, PSNR = 24.5]

These images have been removed

Figure 1.12 Reconstructed image after error being introduced at 1st and 2000th bit positions [Max. Error = 74, MSE = 120.78, PSNR = 27.31]

Figure 1.13 Reconstructed image after error being introduced at 1st and 10000th bit positions [Max. Error = 61, MSE = 116, PSNR = 27.48]

Observations made based on the simulation results illustrate that MSE and PSNR of the reconstructed image is affected due to bit errors. Bit errors in the initial part of the compressed data (within 30 bits in this example) can have significant effect on the reconstructed image. Fig. 1.10 showing the error in the 10^{th} and 30^{th} bit position having catastrophic effect on the reconstructed image. This is due to the fact that the DWT decomposes image into high frequency and low frequency components. As low frequency components of the decomposed image having significant information compared to high frequency components, encoding the decomposed image using SPIHT, low frequency components are arranged first in the encoded bit stream (Fang et al. 2003). Bit errors at the first few bits affect the quality of the image, because more information is stored in low frequency (Cyril 2005). Mean Square Error (MSE), Maximum Error and Peak Signal to Noise Ratio (PSNR) is calculated to estimate the quality of image. Poor quality image has higher MSE and lower PSNR.

The solution to this problem is to request for retransmission of the compressed image. MPEG and JPEG standards recommend use of channel encoding and retransmission schemes based on image quality at the receiver (Turner and Peterson 1992).

The technique proposed in this research does not require the need for error coding and retransmission schemes. This is achieved by use of neural network architectures for image compression and decompression. The objectives of this work are:

1. To prove the effectiveness of neural network architecture in reconstructing the image from the compressed data with noise in the transmission channel.
2. To propose, design, model and implement neural network architectures for image compression and decompression in analog VLSI.

1.4 Motivation to this Work

Vikram Sarabhai Space Centre (VSSC), Trivandrum an entity of Indian Space Research Organization (ISRO), under their RESPOND Scheme awarded the research project entitled "High speed DWT architectures for image compression for launch vehicle applications". The objective was to realize image compression algorithms on FPGA working at speed greater than 25 frames per second. This research work finds application in launch vehicles. As the launch vehicle carrying the satellite has to guide the satellite to the corresponding destination along the predefined path traveling at a speed of 11000 meters per second, monitoring this movement is very critical. Once the destination is reached i.e. the required orbit, the launch vehicle should eject satellite and place it into the corresponding orbit. High speed cameras are mounted to capture images to monitor and guide the launch vehicle to follow a trajectory path. This allows real time monitoring. The images are down linked to the base station from the launch vehicle. Since this is in real time and requires feedback from the base station for guidance of launch vehicle and successful launch of the satellite, high speed architectures are required to compress the images captured and down linked to the base station. The system is time critical as suitable action should be invoked observing the images being received at the base station. During transmission of compressed data channel error on the compressed data affects the quality of the decompressed image. One of the objectives was to find out the impact of

bit error on the decompressed data. During the experimentation, synthetic bit errors (change of '1' to '0' and '0' to '1') were randomly introduced at multiple positions in the compressed stream of data. Corrupted bit stream was decompressed to reproduce the images. It was observed that the bit errors on the first 30 bits of the compressed data had significant impact and reconstruction was impossible with the required information. Schemes recommended by JPEG and MPEG standards adopt retransmission of the frames. Also channel coding schemes are adopted to reduce the impact of error on signal during transmission. For launch vehicle applications as they are time critical retransmission of images may not be a possible solution. Hence, this motivated to investigate use of neural networks for image compression, as neural network techniques have been adopted for signal processing applications (Greenhil and Davies 1994, Guan, Anderson and Sutton 1997, Hanek and Ansari 1996, Lee and Degyvez 1996, Matsumoto, Kobayashi and Togawa 1992, Garris, Wilson and Blue 1998, and Paik and Katsaggelos 1993). Fig. 1.15 depicts simple neural network architecture with input and output neurons. As the network is trained with known set of input and outputs, neural network architectures are immune to noisy environment (Lampinen, J., Laaksonen, J. and Oja, E. 1997). Neural network architectures for image processing are flexible, reconfigurable and can work in noisy environment (Mitra and Yang 1999).

Multiple neurons in multiple layers include weight and bias elements. Weight and bias elements of the neuron decide the functionality of the network. Input image fed into the network, is processed by weight and bias elements of the network. Image processing functions like compression, enhancement, edge detection and segmentation can be executed by the neural network based on the type of weight and bias elements (Koh, Suk and Bhandarkar 1995, Kotropoulos, Magnisalis and Pitas 1994, Le, Thoma and Wechsler 1995, Lin, Tsao and Chen 1992, Manjunath, Simchony and Chellappa 1990, Marshall 1990, Ngan and Hu 1999, Opara and Worgotter 1996 and Ozkan, Dawant and Maciunas 1993). Functionality of neural network architecture depends on number of neurons and

neuron function, interconnection of neurons, and number of layers, weights and bias values.

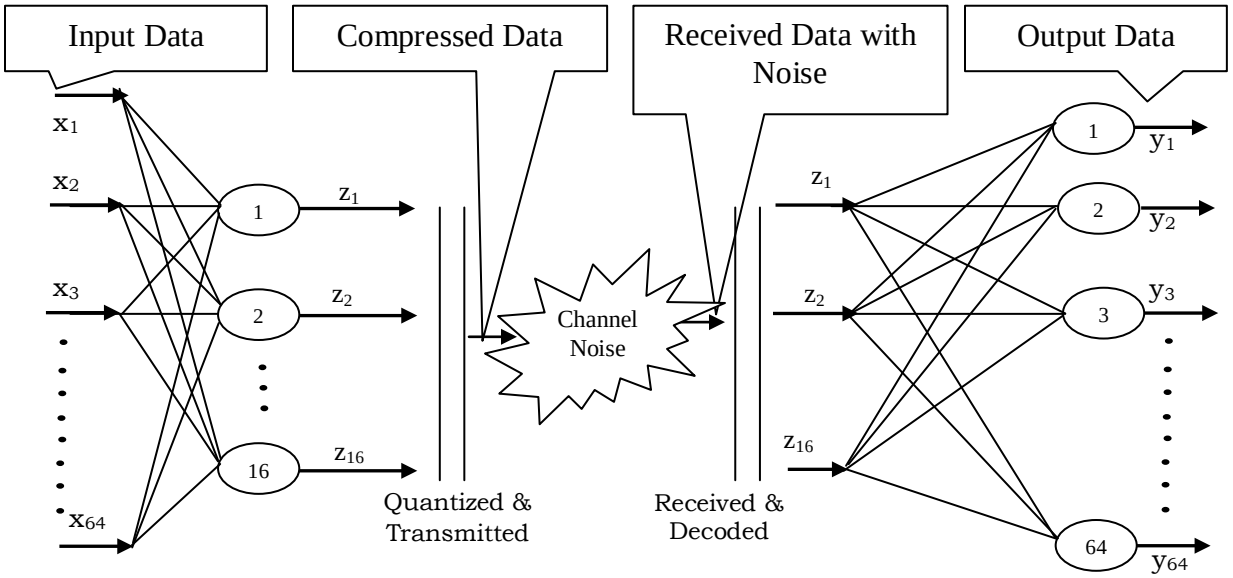


Figure 1.14 Neural networks for image compression

1.5 Need for Analog VLSI

As the input captured from image sensors are analog samples, analog neural network architecture designed using multipliers, adders and network functions are required to process analog samples. The advantage of processing signals in analog domain is that the number of transistors required realizing neural network architecture is much less than digital implementation (Andreou 1992).

1.6 Thesis Contributions

Image sensors capture light intensities of objects with frame rate varying from 30 frames per second to 1000 frames per second, with the time interval between frames are 33 ms and 1 ms respectively. Every frame of image which is in analog form is digitized and compressed using high speed architectures that perform complex digital signal processing algorithms. Hardware implementation of such algorithms consumes power and area, thus increasing the cost of the hardware.

Image conversion from analog to digital, (2-D analog discrete interval to 1-D digital space) introduces delay due to varied reasons. Conventional techniques use powerful transform and efficient encoding techniques for compression and decompression. This research work carried out addresses the issues such as image reordering and image compression together with the proposed architecture eliminating the bottleneck in image acquisition and compression. New techniques for compression and decompression based on neural networks are proposed, realized and validated.

This thesis work investigates the design, modelling, analog VLSI implementation and validation of multidimensional neural network architecture for image compression. The architecture processes image samples stored as current equivalents in 2-D matrix form for testing of the proposed design. Results obtained from this work are compared with digital implementation and software reference models for validation. Techniques and circuits proposed in this thesis are derived based on the references reported in the literature. Validations of the circuits designed and use of them for image compression are carried out based on the results obtained using software and firmware models.

This thesis work demonstrates the need for neural networks for image compression in certain critical application areas like satellite launch applications. The techniques discussed can also be extended to other applications by appropriately choosing network architecture and network functions according to the functionality and the network is trained to realize the functionality. This work uses parameters such as compression ratios, MSE, PSNR and hardware complexities in terms of power, delay and area as a measuring factor to estimate neural network performance. The thesis explores the tradeoff between classical techniques and the neural network techniques for image compression.

VLSI implementation of 2-D architecture for image compression using multilayered neural networks is also discussed in this thesis. This research work proposes two-dimensional multilayered neural network architecture for image compression, thus avoiding rearrangement of 2D image samples to 1D. As the compressed data is in the 2-D

space, two dimensional coding techniques can be adopted to quantize and encode the compressed image. Auto associative neural network uses backpropagation training algorithm for image compression (Sicuranzi, Ramponi, and Marsi 1990, Anthony, D., Taylor, E., D. and Barham, J. (1989), Kohno, Arai, and Imai 1990, Ivan Vilvonic 2006, Omer, M., Farhat, A., Momoh and Salami (2007), Hadi and Mansour 2009). Literature review shows that most of the training algorithm is aimed at training either a single or multilayered neuron. Training a multidimensional network for image compression has not been reported. This thesis work proposes a simplified backpropagation algorithm based training technique for the proposed Two-dimensional Multilayer Neural Network (TDMNN) architecture. Another contribution in this thesis is the design of hybrid architectures for implementation of TDMNN architecture for image compression. The architecture is called as hybrid as it multiplies analog input with digital weights and produces analog outputs. The digital weights (obtained during training) are stored in memory, the hybrid architectures perform multiplication of analog inputs with digital weights. Hybrid architecture is reconfigurable to realize different image processing functionalities by changing the digital weights of the network.

Software reference model for the proposed TDMNN architecture is designed, modeled and simulated using MATLAB. The reference model is trained using image data sets. The weights and biases for the network after training are used to model the compressor and the decompressor. The network is analyzed for its performance using multiple data sets.

1.7 Assumptions and Terminologies

This research work has the following assumptions made:

- Image sensors are not interfaced to the proposed hardware for image compression, instead input reference model equivalent to the natural image is created using MATLAB for software simulations and Cadence Virtuoso for hardware simulations. Test images provided by the standard software are used for

simulation purpose assuming that the properties of these images are identical to a natural image captured from the camera.

- CMOS Technology files for hardware design and verification of the proposed architecture are of industry standard, as they have been obtained from reliable sources www.cadence.com.
- EDA tools for simulation and verification of the circuit behavior are as per the industry standards and are considered as signoff tools. MOS transistors used for realization of the proposed network have been characterized for its performances and have assumed to meet the required functionality. Any other assumptions made within this thesis have been indicated at appropriate places.

1.8 Hypothesis

Image compression using neural networks have been taken as research subject by many of the scientists and engineers. Many new algorithms, architectures, implementation techniques have been extensively reviewed for image compression (Jiang 1999). Literature used as reference for this thesis report multilayered linear or nonlinear network for image compression. Analog VLSI implementation of neuron for general signal processing applications has been reported in the literature, and has been successfully adopted for image segmentation and edge detection. The hypothesis is TDMNN architecture proposed for image compression eliminates the need for digitization of image and reordering of the image to 1-D hence increases the speed of image compression and reduces hardware complexity. This hypothesis will be proved in this thesis by designing the TDMNN architecture using analog building blocks, training the network for different test images evaluating the performances of the network for image compression and optimizing the hardware for area, power and speed performances.

The network performance is compared with DWT-SPIHT technique for image compression. The comparison is only based on the images used for training the network.

Images that were not part of training sets are also used to analyze the network performance.

1.9 Thesis Overview

The thesis is organized as follows. Chapter 2 presents an in-depth literature analysis of neural network for image compression and analog VLSI implementation of neural network architectures. Chapter 3 formally describes the problem statement, methodologies and resources required for carrying out objectives of the proposed work. Chapter 4 presents the software reference model development of TDMNN architecture. Design and analysis of TDMNN model is discussed in detail. Performance of the network based on MSE, PSNR, and Compression ratio is presented. Error analysis and noise analysis is also discussed. Variations in network parameters such as, input size, number of layers, transfer function and training iterations are discussed. Adaptive TDMNN architecture is proposed and results of this architecture is compared with TDMNN and DWT-SPIHT technique. Chapter 5 discusses the hardware implementation of the TDMNN architecture. New architecture for realizing hybrid multiplier is designed and analyzed. Building blocks of the design are integrated to design the neuron model. Detailed discussion of the results obtained based on the experimental work carried out in this research, is presented in this chapter. The thesis concludes in chapter 6 by presenting limitations and directions for further extending this work for video signals.

Appendix – A discusses backpropagation algorithm, Appendix – B presents FPGA implementation of neural network architecture, Appendix – C presents discussion on Entropy, Appendix – D presents the SPICE codes for the designed analog neural network architecture.

Chapter 2 - Neural Networks for Image Compression: Literature Review

Neural networks, image compression, analog VLSI are three unique and distinguishable domains. Image processing is an application, neural network a technique and analog VLSI an implementation domain. This research work is a combination of all the three domains. Neural network architectures are developed for image compression and decompression. The proposed architectures are implemented using analog VLSI circuits. Thus the literature survey in this chapter has two sections. Exhaustive literature review is carried out on neural network architectures for image compression. Secondly, survey on neural network architectures for VLSI implementation is presented. Independent literature analysis is carried out in all the above mentioned topics and a cohesive approach is made to build hybrid architecture.

2.1 Image Compression Using Neural Networks

The study of image compression methods has been an active area of research since the inception of digital image processing. Since images can be regarded as two-dimensional signals with the independent variables being the coordinates of a two-dimensional space, many digital compression techniques for one-dimensional signals can be extended to images with relative ease. As a result, a number of approaches to the problem are well established. Traditional techniques that have already been identified for data compression include (Iain 2002):

a) Predictive Coding

b) Transform coding and

c) Vector Quantization

Traditional techniques for image compression, have been successfully implemented, it is also found that some of the more recent techniques for data compression using artificial neural networks have been reported but commercially are not viable (Russo, Real 1992). Artificial Neural Networks (ANNs) have been applied to

many problems, and have demonstrated their superiority over traditional methods when dealing with noisy or incomplete data. One such application is for image compression. Neural Networks seem to be well suited to this particular purpose, as they have the ability to preprocess input patterns to produce simpler patterns with fewer components. This compressed information (stored in a hidden layer) preserves the full information of a given image. There have already been an exhaustive number of papers published applying ANNs to image compression (Costa and Fiori 2001, Dony and Haykin 1995, and Jiang, W.W., Kiang, S.Z., Hakim, N.Z. and Meadows, H.E. 1993).

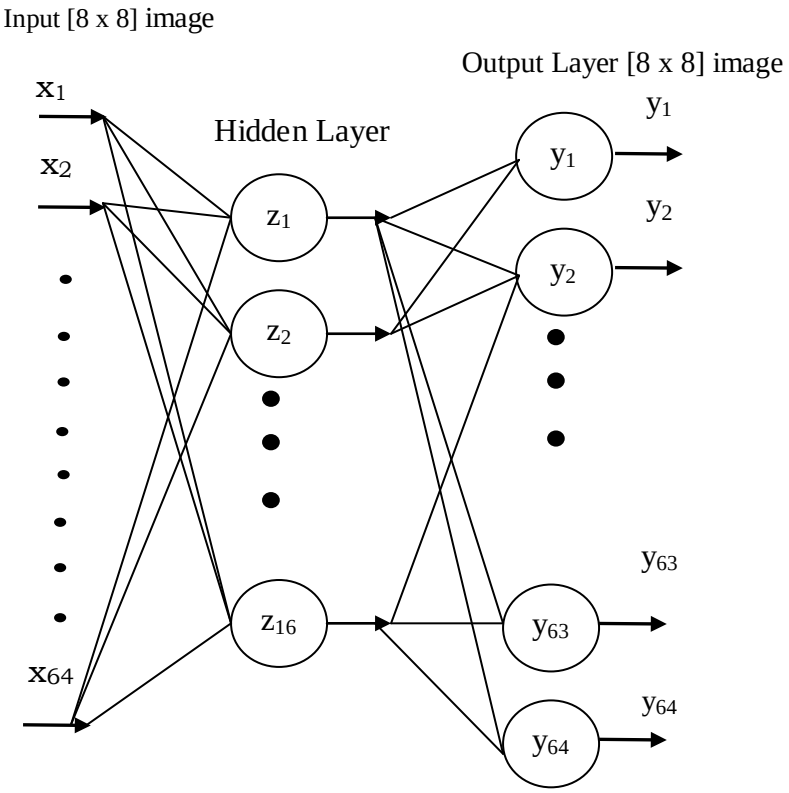


Figure 2.1 Neural network architecture for image compression

The basic architecture for image compression using neural network is shown in Fig. 2.1. The network has input layer, hidden layer and output layer. Inputs from the image are fed into the network through the input layer. The input to the network is the

original image and the output obtained is the reconstructed image. The output obtained at the hidden layer is the compressed output.

The network is used for image compression by breaking it in two parts as shown in the Fig. 2.2. 64 inputs in the input layer are processed to 16 outputs by the hidden layer. The transmitter encodes and transmits the output of the hidden layer (z_1, z_2 …….z_{16}) values instead of (x_1, x_2 …… x_{64}) values of the original image. The receiver decodes the received 16 inputs and reproduces 64 outputs (y_1, y_2 …… y_{64}). Since the network is implementing an identity map, the output at the receiver is an exact reconstruction of the original image. The compressed data is quantized and coded into bit stream.

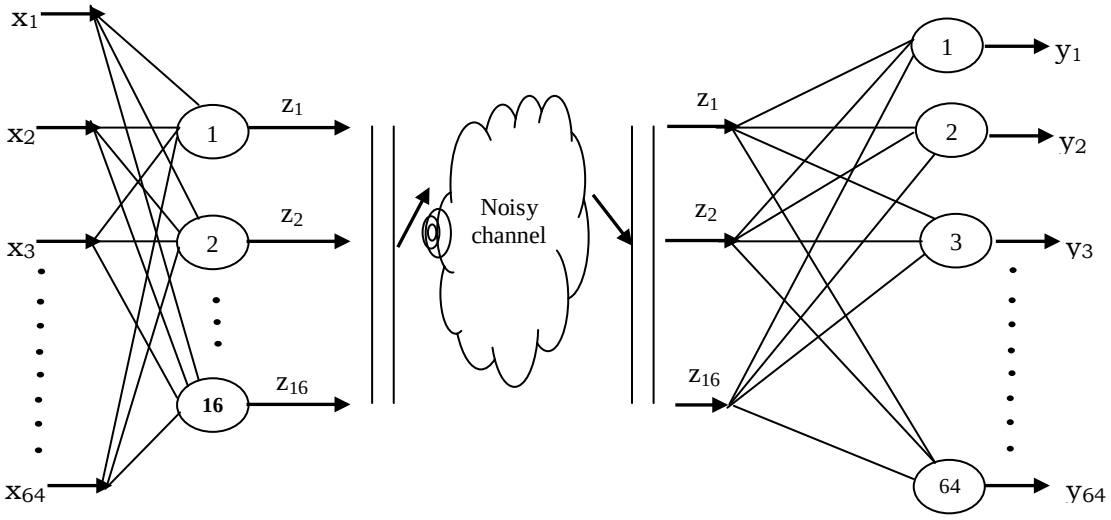


Figure 2.2 Trained neural network to minimize channel noise

Fig. 2.3 shows compression and decompression block diagram using neural network. The compressor block shown in Fig. 2.3(a) consists of 64 data samples that are mapped to 16 data samples. Fig. 2.3(b) shows the decompression unit, which maps 16 input data samples to 64 data samples, thus reconstructing the input data. Appropriate weight and bias elements are required to be determined for the compressor and decompressor unit. Identifying appropriate weights and biases for a given input image is carried out during training phase. Multiple images are used to train the network.

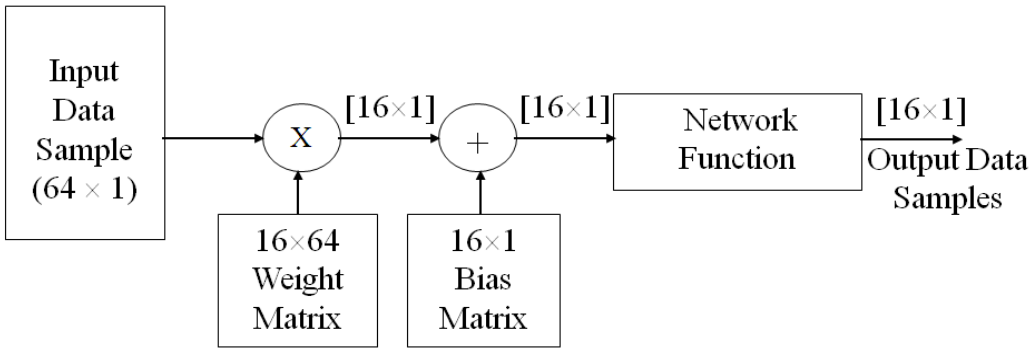


Figure 2.3 (a): Compression block diagram

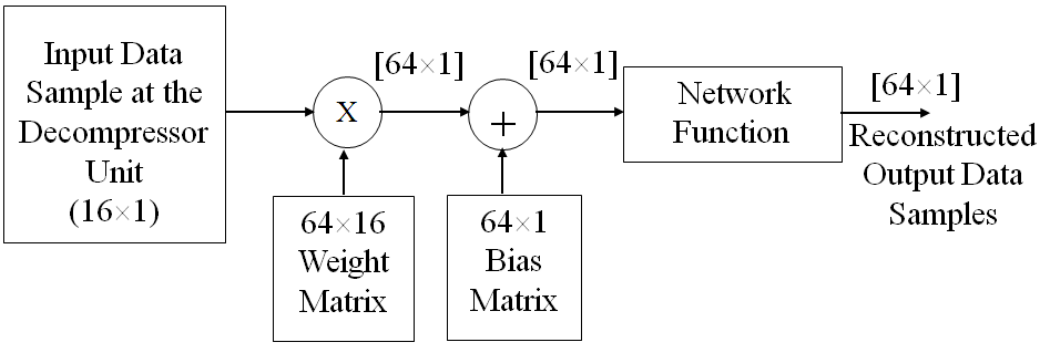


Figure 2.3 (b): Decompression block diagram

Figure 2.3 Compression and decompression block diagram using neural network

The network needs to be trained to reproduce the desired targets, optimum weights and bias elements for the compressor and the decompressor unit is determined during the training phase. A detailed discussion on neural network training is presented in Appendix - A.

As shown in Fig. 2.3, 64 data samples at the input of the compressor are mapped to 16 data samples at the output, and at the receiver section 64 data samples are reconstructed from 16 data samples available at the input of the decompression unit. For the network to process, data inputs are provided in a column matrix. The image which is two dimensional having N rows and N columns need to be rearranged into a column matrix, as the neural network accepts only column inputs. The rearrangement of input

image is to divide the image matrix into multiple blocks of fixed size. Each sub-block is reorganized into column matrix; all the sub-blocks that are rearranged into column matrix are combined into one matrix and are set as input. The same matrix is also set as the target (Becker and Plumbley 1996). Multiple images with varying properties are considered for training, so that the network learns to reconstruct the required image once it is trained.

Fig. 2.4 illustrates the rearrangement process which is required to set the input and targets for the network. Training data sets are produced from the image by extracting small (n x n) blocks of the image. Image of size [N x N] is first sub-divided into (n x n) sub-blocks, each of the (n x n) sub-blocks are further rearranged into (m x 1) elements, where $m = n^2$. For example, an image of size [32 x 32] is sub-divided into (8 x 8) blocks of sub-images. There are 16 sub images of size (8 x 8) in [32 x 32] image. Each of the sub-blocks are rearranged to (64 x 1), hence for the [32 x 32] image, after rearrangement the matrix size is [64 x 16] as shown in Fig. 2.4. If there are 10 images of size [32 x 32], each of them are rearranged into [64 x 16]. All the 10 images are combined into a training data matrix of size [64 x 160]. The network gets trained using the rearranged image matrix. The network shown in Fig. 2.3 processes one column at a time from the rearranged matrix. As illustrated in Fig. 2.3, (64 x 1) input data is multiplied with a weight matrix of size [16 x 64] that compresses input into (16 x 1) matrices. (16 x 1) data is added with a bias element and passed through network function which is either linear or nonlinear. The compressed data (16 x 1) matrix is decompressed to (64 x 1) at the receiver by a weight matrix of size [64 x 16] and further biased and passed through network function. The decompressed output is rearranged into its original form. The error between the original image and the decompressed image is computed. Until the error is minimized the network is trained a detailed discussion on network training is discussed in Appendix - A.

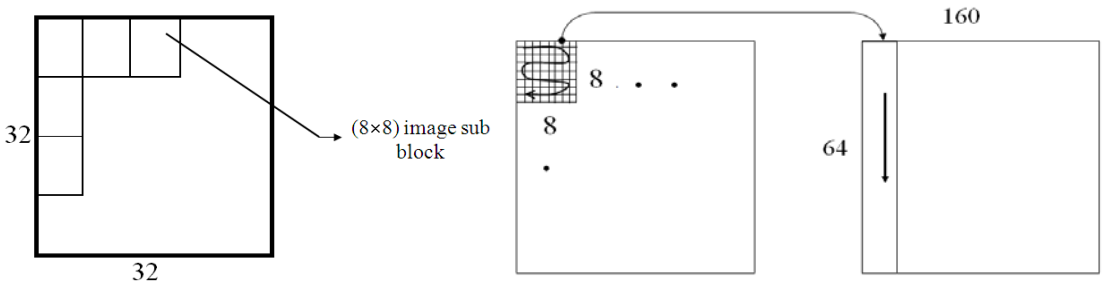


Figure 2.4 Image segmentation to sub-blocks and reordering

2.2 Literature Review on Neural Networks for Image Compression

Oja (1982) proposed simple neural network architectures that can perform image compression based on principal component analysis. Since then several algorithms and architectures for neural network based image compression have been proposed. Oja (1982) proposed a one-unit learning rule to find the first principal component direction vector, i.e. the first eigenvector, such that:

$$Z = W * X \tag{2.1}$$

Where, Z denotes the output of the linear neuron, W denotes the weight element of the neuron and X denotes the input. Oja proved that the weight vector W will asymptotically converge to the first normalized Eigen vector.

Sanger T. D. (1989) extended the work of Oja (1982) to compute more than one Eigen vector leading to principal components. In other words, Oja (1982) had used one neuron to find one Eigen vector representing the principal component of the image matrix X. Sanger extended the single neuron to a network of multiple single linear networks to compute many Eigen vectors representing the principal components of the matrix X. The basic principle of image compression using neural network is to identify the weight matrix of the trained neural network architecture, where the rows of the weight matrix are the principal components of the matrix X. For performance evaluation, Sanger (1989) implemented the algorithm using (8 x 8) input blocks and an output of (8 x 1) vectors. The network was trained using (512 x 512) image using non-overlapping blocks. Mean square error (MSE) of 0.043 at 0.36 bits per pixel (bpp) was achieved on images that

were used for training. Retaining the weight matrix, image that was not part of the training set were compressed and decompressed. MSE of 0.023 was achieved at 0.55 bpp. This led to the generalization of neural network architectures for image compression. Multiple images need to be used as training samples to ensure higher performance of the network in terms of MSE. Sanger's method uses only feed forward neural network architecture. Cottrell and Munro (1988) have proposed direct neural network architecture for image compression. They view the problem of image compression as a mapping issue, where (8 x 8) (arranged as 64 x 1) input is mapped to (64 x 1) outputs at the output layer, the compression is achieved by having 16 hidden layers. In such scheme all the neurons are fully connected. In their work backpropagation training technique was adopted to train the network to obtain the weight matrix representing the principal components of the image. Each sub-block of the image is presented at the input and output of the network simultaneously, and the weights of the hidden layers and output layer are adjusted using backpropagation algorithm. Linear networks were used and were shown to produce comparable results to nonlinear networks. Images tend to span the first M principal components of the image, where M is the number of hidden layers. Sonehara *et al*. (1989) propose a three layer Backpropagation Neural Network (BPNN). The compression is achieved using three layer network the input layer, hidden layer and output layer. The hidden layer has less number of neurons compared to the input and output layer, which achieves compression. In their method the original image was divided into sub-blocks and fed to the input layer, the output layer restores the original sub-block. This implementation was done on the NCUBE parallel computer and the simulation results showed that this network achieved poor image quality. Carrato and Ramponi (1991) modified the direct architecture proposed by Cottrell and Munro (1988) by introducing nonlinear transfer functions. It is reported that nonlinear network have higher compression ratios than linear network. Instead of mathematically analyzing the performances of nonlinear network, their work considers use of multiple image data samples for training the network built using nonlinear functions. Based on the results

obtained the performances are evaluated. Definite proof for use of nonlinear network for better image quality is not reported in this work. Mougeot, Azencott and Angeniol (1991) prove the nonlinear network proposed by Carrato and Ramponi (1991) suffer from poor quality, but achieves better compression. In order to improve the quality metrics of the decompressed image, they used overlapped block samples from the image for training and simulation. This achieves improvement in quality and compression at the cost of training time. Carrato (1992) has reported the use of neural network for image compression which performs better than transform coding techniques. Direct neural network architecture shown in Fig. 2.5 with linear functions is used to prove his claim. Backpropagation neural network was used for training the network for image compression and decompression. The input layer and output layer are fully connected to the hidden layer.

Image compression is achieved by training the network in such a way that the coupling weights, $\{w_{ji}\}$, scale the input vector of N-dimension into a narrow channel of K-dimension (K < N) at the hidden layer and produce the optimum output value which makes the quadratic error between input and output minimum. In accordance with the neural network structure shown in Fig. 2.5, the operation of a linear network can be described as in equation (2.2)

$$z_j = \sum_{i=1}^{N} w_{ji} x_i, \; 1 \le j \le K \tag{2.2}$$

Where, $0 \le x_i \le 1$ denotes the normalized pixel values for grey scale images with grey levels {0 to 255} and equation (2.3) for decoding,

$$y_i = \sum_{j=1}^{K} w_{ij} z_j, \; 1 \le i \le N \tag{2.3}$$

The reason for using normalized pixel values is due to the fact that neural networks can operate more efficiently when both their inputs and outputs are in the range {0 to 1} Carrato (1992).

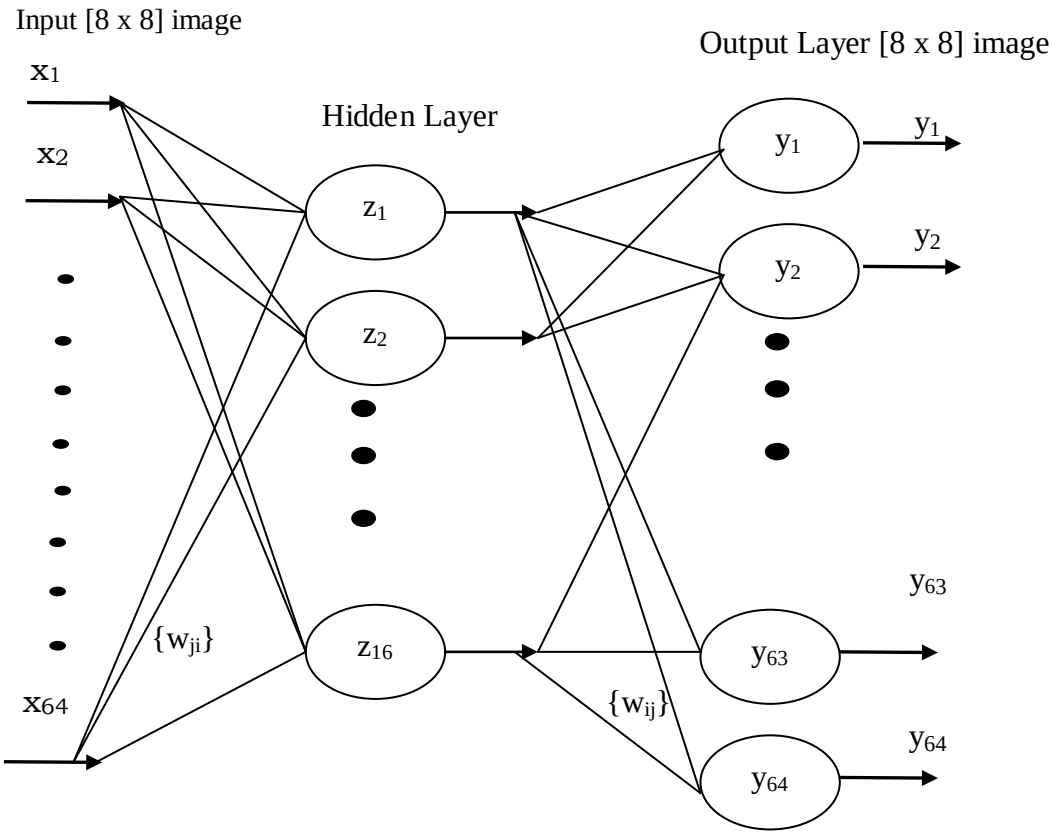


Figure 2.5 Back propagation based neural network architecture

Neural network algorithms and architectures for image compression exists in many types such as Multi Layered Perceptron (MLP) (Qiu, Varley and Terrel 1993), Hopfield (Lin and Liu 1999), Self Organizing Map (SOM) (Amerijckx, Legaty and Verleysen 2003), Learning Vector Quantization (LVQ) (Pavlidis *et al.* 2001) and Principal Component Analysis (PCA) (Costa and Fiori 2001).

Namphol, Chin and Arozullah (1996) propose hierarchical neural network architecture in order to improve the compression ratio. Image is classified into multiple sub-images, based on these sub-images the network gets trained. This approach achieved better compression and also improved quality of reconstructed image. The number of hierarchies and number of layers are decided based on the image quality. The idea was to exploit correlation between image pixels and correlation between blocks of pixels in the

inner hidden layer and outer hidden layer respectively. From the input layer to the hidden layer and from the hidden layer to the output layer, local connections are designed which have the same effect as several fully connected neural sub-networks. The total number of neurons for the hidden layer and the output layer is the same as that of the hidden layer and the input layer, respectively. Benbenesiti (1997) modified the multilayer network proposed by Namphol, Chin and Arozullah (1996) to achieve better quality. Inputs, weight elements and outputs are normalized to the range 0 to 1. This results in improvement in performance of neural network architecture in terms of higher compression and image quality.

Jiang (1999) discussed various architectures for image compression using neural networks. Jiang (1999) classified neural network architectures into three different categories. These include direct neural network architecture, traditional neural network approaches and indirect neural network approaches. In his work architectures are compared based on their performances, the major bottleneck is the training time required for the neural network architecture. Parallel processing with programming capability is recommended for hardware implementation of neural network architectures for image compression. Implementing neural network architectures on hardware is one of the major challenges identified by Jiang (1999).

Benbenesiti et al. (1999) have also used linear network for compression, with backpropagation training algorithms, and reordering of the image to 1-D. They have also implemented nonlinear network based image compression and have obtained better results, with 70% improvement in compression ratio. Lewicki and Olshausen (1999) have used multilayer networks with nonlinear components to achieve compression and have proven that nonlinear networks are better than linear networks in compression with improved image quality, but nonlinear networks increases complexity and training time. In their work, multilayer networks have only nonlinear nodes. A two layer perceptron network forms unbounded, convex region in the space spanned by the inputs. A three layer perceptron network forms arbitrarily complex decision regions. A three layer

perceptron network could thus be used to create any continuous likelihood function required in a classifier. The above discussion on the classification and approximation capabilities of the multilayer neural network is based on the fact that all the layers are trained simultaneously. It is important to emphasize here that training each layer separately, does not share the proven capabilities of the standard multilayer neural network. In their work, a four layer nonlinear network is proposed to further enhance the compression ratio. Using four layer nonlinear networks, MSE was 47.36 where as for two layer linear networks MSE was 86.49.

Ivan Vilovic (2006) uses nonlinear networks for image compression. He has taken images in to overlapping sub-blocks for training. Training time is more in this case. Circuit complexity increases with nonlinear elements. He has compared the results of NN with JPEG technique. Feed forward networks with backpropagation technique are adopted for the analysis. Sub-divided images of various sizes such as (4 x 4), (8 x 8) and (16 x 16) have been considered and transformed into column matrix as discussed earlier and is used for compression and decompression. Results obtained based on his work is shown in Fig. 2.6, SNR for JPEG is found to be better compared with NN based technique. Use of new algorithms and techniques to overcome deficiencies of NN technique is recommended by Ivan Vilovic (2006). Most of the results reported do not show significant advantage in terms of SNR over JPEG.

Steven and Mario (1998) have realized neural network for image compression using Field Programmable Gate Array (FPGA). They used backpropagation technique for training the network. Linear networks have been considered for compression of images. Linear neural networks are well suited for FPGA implementation as the complexity of the hardware is reduced. The uniform nature of the model permits a single circuit to be configured and used to calculate the outputs of the network. Due to the complexity involved in mapping architecture on FPGA, only a single neuron is realized.

This image has been removed

Figure 2.6 SNR vs. CR for NN and JPEG codec (Ivan Vilovic 2006:26)

From the review carried out it is significant that neural network approaches for image compression have been extensively used and many authors have achieved good performances in terms of MSE and PSNR. Some of the common factors among the reported architectures are: Image compression using neural network is achieved using multilayer network. Neural network architecture consisting of input layer, hidden layer and output layer that are fully connected are used for compression. Compression is achieved by selecting the hidden layer size less than the input and output layer. Nonlinear and linear functions have been used as network functions. Training of the network with multiple images achieves better results. Images are sub-divided into sub-blocks (Overlapping and non-overlapping) and rearranged prior to training. Hierarchical networks are also used. All the work reported in the literature have realized and proven the network performances based on software models. From the above discussion, the following are conclusions:

1. Multilayer neural networks have been adopted and recommended by many of the work reported.
2. Backpropagation algorithms have been used by most of the work reported for training the network.

3. Direct feed forward backpropagation neural network architectures are simple are successfully used for compression and decompression.
4. Training of the network is a very important phase; image sets selected also plays an important role in identifying optimum weight elements for the network.
5. The transfer function used in neural network classifies the network as linear or nonlinear network. Results presented in most of the literature are inconclusive on the performances of linear and nonlinear networks for compression and image quality.
6. There is no mathematical proof showing the efficiency of nonlinear network over the linear network for image compression.
7. SNR is observed to be less for neural network technique in comparison with conventional JPEG as per Ivan Vilovic (2006).
8. Most of the NN architecture work reported in the literature is based on image compression using multilayered network and hierarchical network Jiang (1999).
9. Although significant work has been done towards neural network development for image compression, and strong competition can be forced on conventional techniques, it is premature to say that neural network technology can provide better solutions for practical image coding problems Jiang (1999). For real-time applications, efficient hardware's are required to implement NN architectures.
10. Coordinated efforts world-wide are required to assess the neural networks developed on practical applications in which the training set and image samples should be standardized. In this way, every algorithm proposed can go through the same assessment with the same test data set as per Jiang (1999).
11. The full potential of neural network approaches will not be realized until they are implemented in their true parallel form. Most of the implementations used in the above research have been based on simulations on computers. With the development of VLSI implementations for many neural network architectures, the speed for both training and coding will dramatically increase.

12. Hardware implementation of NN architecture for higher image sizes is not reported, most of the techniques reported in the literature are only software simulation and the hardware implementation is only for a very small image size.
13. Flexibility in compression ratio is one of the major limitations in NN based architectures for image compression.
14. All the neural network techniques reported use image samples that are reordered to column matrix from the original sub-image blocks. Processing images or compressing images without reordering is not reported in the literature.

From the above observations made, major and specific gaps in the literature are:

1. Image reordering is carried out and there are no claims reported on two-dimensional multilayer neural network architecture for image compression.
2. Nonlinear and linear neural networks have been used for image compression and decompression. However, their performances when compared with conventional technique are not very significant Ivan Vilovic (2006).
3. Highly powerful and computationally intensive parallel architectures are required for neural network architectures to be realized on hardware.
4. New techniques and architectures are required to improve the SNR of NN based techniques over traditional techniques for image compression.

Given that there is a viable solution for image compression using neural networks, the next step is to build the neural network hardware. The hardware implementation of any concept should be cost effective, feasible and reliable. Also, the hardware should be area, power and speed efficient. A fully digital approach using RISC processor is one possible solution. Such processors are designed to execute simple instructions, preferably one instruction every cycle. For realizing the massively parallel and computational intensive neural network architecture analog VLSI design is considered in this work. VLSI technology is well matched to realize neural network architectures for two basic reasons:

1> VLSI technology permits implementation of large number of identical, concurrently operating neurons on single chip, thus exploiting the parallelism property required for neural networks.

2> Well defined arithmetic operations and relatively small and reliable circuits greatly simplify VLSI implementation complexities.

Next section discusses analog VLSI implementation of neural network architectures.

2.3 Artificial Neural Network

Biological neurons are artificially modeled as shown in Fig. 2.7. The basic building blocks of a single neuron are multipliers, adders and network function. Other than these building blocks data storage elements are required to store the input, output, weights and bias elements. The neural network shown Fig. 2.7, can be classified in terms of their implementation into three categories: Digital, Analog or Hybrid. Digital network is realized using logic gates. Network inputs, weight, bias and output are all represented in binary format.

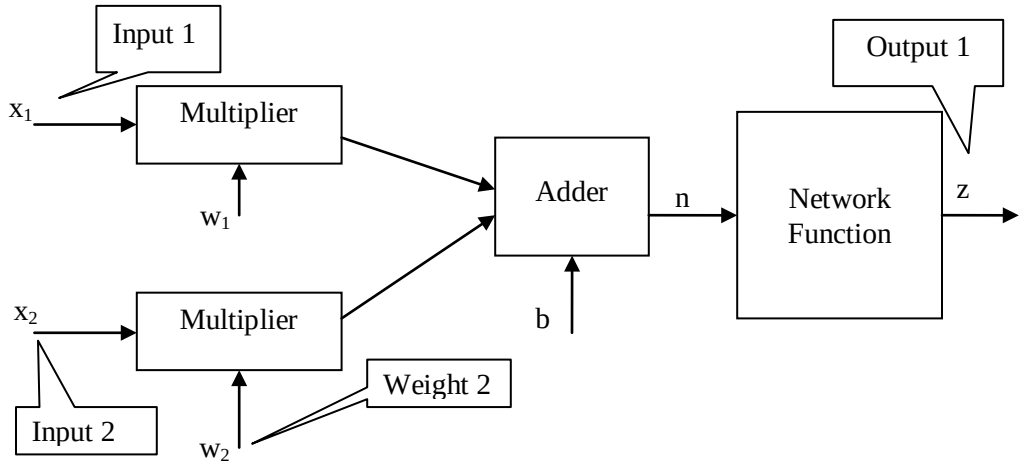


Figure 2.7 Two layer neuron with 2 hidden layers and 1 output layer

In analog network, transistors and passive components are used to realize the network. Input, weight, bias and output are all in analog form. In hybrid network a combination of digital blocks and analog blocks are used to realize the network. Hybrid

also implies that, the input, weight, bias and output can be in either analog or digital. The implementation of the neural network architecture (NA) in all these categories requires learning capability to be integrated in the design. This learning capability or learning rules are based on the mathematical algorithms that are unique for a given network. The focus of this research work is to implement the neural network architecture with backpropagation learning/training algorithm for image compression realized using hybrid technique.

Neuron comprises of multiplier and adder along with the network function (Jiang 1995). Fig. 2.7 can be expressed mathematically as in equation (2.4) and equation (2.5),

$$n = x_1 * w_1 + x_2 * w_2 + b \tag{2.4}$$

$$z = f(n) \tag{2.5}$$

Where, z is the neuron output and n is the intermediate output for the inputs x and neuron weights w. Bias b is optional.

Training of the network to realize a given functionality is achieved by setting a target for a given input. The initial weights and bias of the network is assumed. The input vectors are presented to the network, based on the initial weights and bias elements the outputs are computed. The error between the target and the obtained output is used to update the weight and bias elements. This process is continued until the error converges to minimum. The error is backpropagated to update the weight and the bias elements hence the algorithm is named as backpropagation. The use of analog computation is attractive for neural network due to its compactness, potential speed, absence of quantization effects and reading analog samples without the need for digital conversion. The use of digital techniques on the other hand is preferred due to its robustness, easy transmission and regeneration, simplicity and flexible. Hybrid circuits are a combination of both digital and analog blocks has the merits of both analog and digital implementation. In this research work, new hybrid schemes are proposed for image compression using neural network architectures. Next section, reviews various schemes

for realizing neural network architecture that forms a platform to develop new hybrid architectures for image compression.

2.4 Neural Network Architectures

The neural networks are broadly classified as feed forward and recurrent networks (Bose and Liang 2006). In Fig. 2.8 (a) the feed forward network is shown. The first layer is the input layer, from which the neuron gets the input from sensors or data storage units. The last layer is called the output layer. The intermediate layer is called the hidden layer. As can be seen there is no feedback of the outputs again to the inputs. In Fig. 2.8(b) a recurrent neural network is shown. They have a kind of direct cycles in their connection graphs. As shown in Fig. 2.8(b), the outputs are going back to the output layer as inputs. The outputs are recurrent, as they have the effect on the next outputs. In this thesis only the feed forward networks are considered and designed using hybrid architecture.

Figure 2.8(a): Feed forward network Figure 2.8(b): Recurrent network

Figure 2.8 Different types of neural network architectures (Bose and Liang 2006)

2.4.1 Single Layer Neuron

In Fig. 2.9 single neuron is shown. The x input matrix multiplied with the weights (w) and modified using bias (b) is summed (n) and passed through the network function f, mathematically it can be shown as in equation (2.6) and equation (2.7),

$$n = \sum_{i=1}^{R} x_i w_{1i} + b \qquad (2.6)$$

$$z = f(n) \qquad (2.7)$$

Where, n is the summed output of the multipliers and z is neuron output.

This image has been removed

Figure 2.9 Single neuron (Bose and Liang 2006)

In Fig. 2.10 a single layer of neuron network is shown. The input matrix x is now connected to layer of neurons. Each neuron receives x inputs and is processed to one output from each neuron. Each output of neuron is further passed through a network function to compute *s* number of outputs as shown in Fig. 2.10. The equation (2.6) and equation (2.7) are modified to equation (2.8) and equation (2.9) for single layer neuron network and is given as,

$$n_j = \sum_{i=1}^{R} x_i w_{ij} + b_j \qquad (2.8)$$

$$z_j = f(n_j) \qquad (2.9)$$

2.4.2 Multiple Layer of Neurons

The set of single layer neurons connected with each other is called the multiple layer neurons, as shown in the Fig. 2.11. The inputs (input layer) are connected to the layer 1 which in turn is connected to the layer 2 which in turn is connected to the output layer 3. Layer 2 is the hidden layer, and layer 3 is the output layer. Inputs (X) is taken into hidden layer for processing, the output of hidden layer is further processed by the output layer to produce outputs (Z) of the network. As the input is processed by more than one layer of network, this architecture is called as multilayered neural network architecture.

Figure 2.10 Single layer of neuron (Bose and Liang 2006)

These images have been removed

Figure 2.11 Multiple layers neural network (Bose and Liang 2006)

2.5 Backpropagation Algorithm

The essence of the neural network lies in the way the weights are updated. The updating of the weights is through a definite algorithm. In this thesis, Backpropagation (BP)

algorithm is modeled and implemented. The algorithm is applied for the supervised learning that is a desired output will be applied to neural architecture. The target is represented as d_i (desired output) for the i^{th} output unit. The actual output of the layer 2 is given by z^2_i.

Thus, the error or cost function is given by equation 2.10 **(**Bose and Liang 2006)

$$E = \frac{1}{2}\sum_{i=1}^{s}(z_i^2 - d_i)^2 \qquad (2.10)$$

This process of computing the error is called a forward pass. How the output unit affects the error in the i^{th} layer is given by differentiating equation (2.11) by

$$\frac{\partial E}{\partial z_i^2} \qquad (2.11)$$

The equation (2.12) can be written in the other form as

$$\partial_i = (z_i^2 - d_i)d(z_i^2) \qquad (2.12)$$

where, $d(z_i)$ is the differentiation of z_i. The weight update is given by

$$\Delta w_{ij} = \eta \partial_i z_i^1 \qquad (2.13)$$

Where z^1_i is the output of the hidden layer or input to the output neuron and η is the learning rate (Bernabe and Barranco 1992). This error propagates backwards from the output to the input. The δ for the hidden layer is calculated as

$$\partial_{hiddenlayer} = d(z_i^1)\sum w_{ij}\partial_i \qquad (2.14)$$

Weight update for the hidden layer with new δ, is computed using equation (2.14). Equation (2.10) – (2.14) depend on the number of the neurons present in the layer and the number of layers present in the network. Appendix – A discusses backpropagation algorithm.

2.6 Neural Networks in Analog VLSI

Neural network described so far is a multilayer feed forward network with backpropagation algorithm. The basic building blocks in the neural networks are the

multipliers, summers and tan-sigmoid function. Multiplier is an essential component in the neural network. Commonly known as the synapse, the multiplier performs the multiplication of the input with the weight function. Reviews of different analog VLSI architectures for neural network are presented in this section.

2.6.1 Modular T-Mode Design

Bernabe Linares (1992) came up with the modular transconductance-mode approach shown in Fig. 2.12. x_1, x_2 ….. x_N are the inputs that are multiplied by the network weights w_{1i}, w_{2i}…….w_{Ni}. Current output of transconductance amplifier is summed up at the node x_i and is processed by the translinear amplifier (f()) as shown in Fig. 2.12. As the neuron size increases, the transconductance amplifiers connected to the node x_i also increases.

The node current I_i is a function of x_N and w_{Ni} represented mathematically as in equation (2.15),

$$I_i = x_1w_{1i} + x_2w_{2i} + x_3w_{3i} + \ldots\ldots\ldots\ldots\ldots x_Nw_{Ni} + I_1 \quad (2.15)$$

Where, x_1, x_2 …x_N are the inputs, w_{1i}, w_{2i} ....., w_{Ni} are the weights, I_1 is the bias current.

This image has been removed

Figure 2.12 Neuron interconnections using transconductance devices (Bernabe 1992)

Fig. 2.13 shows MOS transistor schematics for transconductance amplifier and translinear amplifier. Gilbert cell is used as transconductance amplifier.

VSS

Figure 2.13(a): Transconductance amplifier

These images have been removed

Figure 2.13(b): Nonlinear circuit or translinear amplifier

Figure 2.13 Building blocks of analog neural network (Bernabe and Barranco 1992)

The output current of the transconductance amplifier is Iout = g_m x_i $w_{Ni,}$ where x_i is the input to the neuron, gm is the transconductance of the amplifier and w_{Ni} is the network weight. As shown in Fig. 2.13(a) for a given neuron input x applied at the gate of transistor M_2 and M_3, w input applied at the gate of transistor M_5 controls the current flow from V_{DD} to V_{SS}. The nonlinear network function shown in the Fig. 2.12 was

implemented using the circuit shown in the Fig. 2.13(b). This architecture was used to implement the winner takes all network and the Hopfield networks. As the circuit requires resistors, capacitors, and transconductance amplifier for realization, the circuit is complex, and has less accuracy.

j

Figure 2.14 Neuron in the output layer (Hussein 1997)

Hussein (1997) in his PhD thesis designed neural network architecture with on-chip supervised learning. He designed the multilayer perceptron architecture trained by the Backpropagation algorithm. Fig. 2.14 shows the neuron in the output layer and the neuron in the hidden layer. The block A is activation function. D is the derivative block which performs the derivative of the input. R block generates the error signal that goes to the other synapses to update the weights. The FC block takes in the outputs and calculates the cost function described as in equation (2.5).

Multiplier block is implemented using the operational transconductance amplifier shown in Fig. 2.15(a). The backpropagation algorithm used in the design is the modified version of Vogl Back Propagation (MVBP). The author compared modified Vogl and Vogl backpropagation (VBP) algorithms and concluded that the learning rate of MVBP is faster than VBP. He successfully trained the network for logic functions like XOR, AND, OR gates. It is observed that this circuit supports online training, the complexity of the circuit is due to the number of transistors required to realize the blocks. The circuit for the hyperbolic tangent function is shown in Fig. 2.15(b).

Figure 2.15 (a): Multiplier

Figure 2.15 (b): Hyperbolic tangent function

Figure 2.15 Analog neuron circuit diagram (Hussein 1997)

Roy (1994) in his thesis described the analog neural network with on chip learning. He used the single ended inputs in his design and a modified transconductance amplifier for multiplication purpose as shown in Fig. 2.16(a). The circuit has large linear range. The author used the backpropagation algorithm for the training. The derivative

circuit is also shown in the Fig. 2.16(b). The I_{bump} current in the figure is the derivative output of the Neuron activation function. The author designed a 4-3-2 neural network. He had four inputs and two outputs, which required 2 target signals for training the network. The design was validated for 4 input XOR function and an 8 input XOR function.

Figure 2.16(a): Transconductance amplifier

These images have been removed

Figure 2.16 (b): Bump circuit

Figure 2.16 Amplifier and adder circuit (Roy 1994)

Chun Lu, Bing-xue Shi and Lu Chen (2002) in their paper presented a method of implementing analog accumulator used in the Backpropagation (BP) algorithm. For the usual BP algorithm the parallel approach is shown in the Fig. 2.17. As M increases the hardware complexity for implementation also increases. The authors proposed the

structure where the inputs were given in the sequential manner as shown in the Fig. 2.17. Even if M increases, the hardware complexity remains the same.

Figure 2.17 Neuron circuit (Chun, Bing-xue and Lu 2002)

Circuit realization of the accumulator in Fig. 2.17 is shown in Fig. 2.18. Cells A, B and C are comparators. Clocks clk1 and clk2 are the non-overlapping clocks. Input a from the accumulator and the output b are compared and the error is used to update the weight matrix. Every time new inputs and targets are presented the network needs to be trained and hence is time consuming.

These images have been removed

Figure 2.18 Analog accumulator (Chun, Bing-xue and Lu 2002)

Shai, Cai-Qin, Geiger and Randy (1987) have designed a four quadrant multiplier using Gilbert cell multiplier. The nonlinearity error calculated was less than 0.5% at 75% of full-scale swing. The circuit complexity is four times the simple Gilbert cell multiplier.

Andreas, G. and Boahen, A. (1996) in their paper provided an overview of translinear circuit design using MOS transistors working in sub threshold region. They contrasted the bipolar and MOS sub threshold characteristics and extended the translinear principle to the sub-threshold MOS Ohmic region through a drain source current decomposition.

From the review carried out on VLSI implementation of neural network architectures, the following are the conclusions drawn:

- ⇒ The basic building blocks for a neuron network are multipliers, adders and network functions
- ⇒ Gilbert cell is used for realizing the multipliers in many circuits, transconductance amplifier are also used for realizing multipliers
- ⇒ Most of the reviewed architectures generate current as circuit outputs, this simplifies circuit complexities

2.7 Digital Implementation of Neural Network Architectures

In this section a brief review of digital implementation of neural network is discussed. As the neural network architecture is massively parallel and hence computational intensive, realizing the neural network architecture on processor platforms have their own merits and demerits (Hammerstrom 1992). Dedicated high speed parallel processors are required to realize neural network architectures. In this discussion, Application Specific Integrated Circuits (ASIC) and Field Programmable Gate Array (FPGA) based implementations are discussed. ASICs are the ideal choice when high performance is required. However, long development time is needed and the complexity of the system could become unavoidable. In this kind of designs, an area-flexibility trade-off is found (Kristian 2003). ASICs cannot easily provide the flexibility demanded by NNs (diverse learning algorithms,

reconfigurability, etc.). Consequently, hardware sharing policies have to be used to optimize the hardware resources. Different solutions regarding the implementation of NNs on programmable digital devices (FPGAs and DSPs) can be found in the literature (Backus 1978, Blelloch 1990, Cox and Blanz 1992, Hillis 1985, Auda and Kamel 1999 and Beuchat, Haenni, and Sanchez 1998). In these cases it is needed to employ many of these devices increasing both price and power dissipation. As discussed in earlier sections, the basic building blocks for implementation are the adders, multipliers, transfer functions and memory. Training is another very important factor when working with neural network architectures. The hardware realization should support both training and functionality realization. Hence there is a need for reconfigurability of the hardware.

2.8 FPGA Implementation of Neural Network Architecture

Several architectures based on Field Programmable Gate Arrays (FPGAs) have recently been introduced. These machines have demonstrated a high level of performance for a variety of problems. Recently, several FPGA-based machines have been designed and built. These machines have demonstrated supercomputer-level performance for a variety of computationally intensive problems. In spite of these impressive demonstrations, FPGA-based machines have not found widespread use. One limitation of these machines is their programming environment. For the most part, these machines have been programmed using hardware design tools. While this approach permits the most flexibility and highest performance, it requires that the programmer be a skilled hardware designer. Guccione and Gonzalez (1993) have proposed a more traditional programming model for these machines based on the vector-based data-parallel model of computation. This model takes algorithms described in a high-level C-like language and translates them into high-performance digital circuits.

Backpropagation based neural networks currently stand out as the most popular type of neural network used to date and have been successfully implemented on FPGAs (Omondi and Rajapakse 2006, Pavlitov and Mancler 2004, Blake and McDaid 2005 and

Kwan, 1992). Eldridge (1994) successfully implemented the backpropagation algorithm using a custom platform built out of Xilinx XC3090 FPGAs, called the Run-Time Reconfiguration Artificial Neural Network (RRANN). This architecture could learn how to approximate centroid of fuzzy sets. Results showed that RRANN converged on the training set, once 92% of the training data came within two quantization errors (1/16) of the actual value, also RRANN generalized well as 88% of approximations calculated by RRANN (based on randomized inputs) came within two quantization values. Heavily influenced by the Eldredge's (1994) RRANN architecture Beuchat, Haenni, and Sanchez (1998) developed a FPGA platform, called RENCO-a Reconfigurable Network Computer. As its name implies, RENCO contains four Altera FLEX 10K130 FPGAs that can be reconfigured and monitored over any LAN (i.e. Internet or other) via an onboard 10 Base-T interface. RENCO's intended application was hand-written character recognition. One challenge in implementing the backpropagation on FPGA is the sequential nature of processing between layers. A major challenge is that pipelining of the algorithm cannot occur during training. This problem arises due to the weight update dependencies of backpropagation, and as a result, the utilization of hardware resources dedicated to each of the neural network's layer is wasted. Aaron Ferrucci (1994) and Marcelo H. Martin (1994) built a custom platform, called Adaptive Connectionist Model Emulator (ACME) which consists of multiple Xilinx XC4010 FPGAs. ACME was successfully validated by implementing a 3-input, 3-hidden unit, 1-output network used to learn the 2-input XOR problem. Skrbek (1999) also used this problem to prove that his own custom backpropagation based FPGA platform worked. Skrbek's (1999) FPGA platform, called the ECX card, could also implement Radial Basis Function (RBF) neural networks, and was validated using pattern recognition applications such as parity problem, digit recognition, inside-outside test, and sonar signal recognition.

The type of neural network used in FPGA-based implementations is an important feature used in classifying such architectures. The type of neural network applied depends on the intended application used to solve the problem at hand. Current trends in

this field have shown that there have been very few attempts at implementing modular neural networks on FPGA based platforms. In the past, this can be attributed to the fact that FPGA densities and speeds were inadequate for supporting modular neural networks. However, FPGA densities and speeds have now improved to the point where it's far more feasible to support modular neural networks, and attempts at doing so should be re-visited. The key advantage to the digital approach over analog approach is the ease of design, manufacturability and flexibility. In analog approach, circuit reliability is always a concern; also training the analog network is always a challenge. In digital approach, with software controlling the hardware, both training and functionality is realizable. However, the major disadvantage is that the digital implementation is very time consuming, area hungry and power hungry. Also, digital implementation is driven by the algorithm and the architecture selected.

To measure the computation complexity of digital implementation of neural network architecture, feed forward neural network with 16 inputs, 4 hidden layer and 16 output neurons is designed (Cyril Prasanna Raj, P. and Pinjare S. L. 2005). The network is trained using back propagation technique using multiple training data sets. Optimum weights and biases are identified, Binary Canonic Sign Digit (BCSD) based multiplier and Ripple Carry Adder is designed. HDL model for the 16:4:16 neural network architecture is developed and synthesized using Xilinx ISE. FPGA based implementation of neural network architecture for image compression and decompression is discussed in Chapter 4. In this research work, we address the implementation issues of neural network architectures for image compression based on hybrid architectures.

2.9 Literature Review Summary on VLSI Implementation of Neural Network

From the above discussion, the conclusions drawn in this section based on the literature review further lead to the scope of this research work, addressing the gaps in analog neural networks for image compression. The following are the conclusions that can be drawn:

- Analog neural network requires multiplier, adder and neuron activation function. Training is also similar to the digital network. Analog neuron is able to execute both digital and analog functions similar to the digital network. From the literature analysis it is found that analog neural networks are on par with digital neural networks, secondly analog neurons are faster, as training time is less when compared with digital network training.
- Gilbert cell based transconductance multipliers have been used to implement analog neural network. It requires less number of transistors compared with a typical digital multiplier. Circuit complexities of analog implementation of neural network architecture are much smaller than the digital implementation.
- There is need for two-dimensional multilayer neural network architecture.
- The inputs are analog intensities in spatial domain; the two-dimensional multilayer network should be implemented in the analog domain. Analog implementation minimizes circuit complexity and is faster.
- Two-dimensional multilayer analog neural network architecture should be trained, and the trained weights should be stored for signal processing.
- The network should be used for image compression and decompression.

Next chapter discusses the aims and objectives of the research work, highlighting methods and methodology.

Chapter 3 - Problem Definition

From the literature review carried out in the previous chapter, feed forward multilayer neural network architecture with backpropagation training has been adopted for image compression. Many variations of this architecture have been proposed and realized for improving the performance of the network with respect to image quality. Modifications in backpropagation algorithm have also been suggested to improve the accuracy of the network architecture. A very critical observation made during the literature review is that the network works on one-dimensional image data. Image being 2-D, and captured in the analog form is digitized into N-bit number by row-column read out as discussed in chapter 1. Hence, analog pixel intensities of image captured using charge coupled device can be taken in two-dimensional without converting into one-dimensional digital form. Two-dimensional multilayer feed forward neural networks (TDMNN) architecture for image compression is proposed. This avoids conversion from 2-D to 1-D as being carried out in the conventional and neural network techniques reported. Also, analog data need not be converted to digital data, and hence eliminates analog and digital conversion, which further saves time involved in data preprocessing. Based on the literature review and observations made, the aim and objectives for the research work is formulated and discussed in this chapter. The methods and methodologies required to carry out the objectives is also highlighted. Based on the objectives and methods, two-dimensional multilayer neural network architecture is designed, modeled and implemented for image compression and validated for its performances.

3.1 Aim

To design two-dimensional neural network architecture, implement and optimize for area, power and speed, and validate its performance for image compression.

3.2 Objectives

1. To explore the use of artificial neural networks for image compression with existing architectures in literature.
2. To propose, analyze and validate two-dimensional neural network for image compression.
3. To design and test analog building blocks suitable for realizing the 2-DMNN.
4. To train and optimize analog 2-DMNN for image compression.
5. To optimize the proposed architecture with available architectures for area, power and speed performances.
6. To validate the proposed architecture performance for image compression and image quality.

3.3 Methods and methodologies to carry out the objectives

- Literature review on artificial neural networks and architectures for image compression, analog blocks required for NN implementation, image acquisition, conventional compression techniques, image quality, hardware and software platforms for implementation and compression metrics is carried out by referring journals, conference papers, patents, books and related documents
- Literature review of existing algorithms and architectures for image compression and decompression is carried out highlighting the major limitations and advantages
- Identification of standard image data sets and standard test results is carried out based on literature review
- The gaps in literature are identified for further investigation based on literature review and experimental results
- Pilot studies on available techniques of image compression using NN is carried out

- Software reference models for the algorithms and architecture reported in the literature are developed. The algorithm performance is estimated for image compression in terms of MSE, PSNR and hardware complexities
- Limitations of the neural network algorithms for image compression compared with conventional techniques in terms of MSE, PSNR and maximum error are identified
- Two dimensional neural network architecture for image compression is proposed based on the source of data from camera and its operation to meet 2D compression requirement
- 2-DMNN is modeled using MATLAB and trained using standard image data sets
- Performance of 2-DMNN is estimated and compared with the results of pilot studies
- Mismatches in the results have been considered and suitable modifications is incorporated to obtain better performance
- 2-DMNN is modified based on the results obtained to obtain better performances
- 2-DMNN model is validated using standard image data sets
- Adaptive 2-DMNN architecture is proposed, designed, simulated using MATLAB
- Performances of Adaptive 2-DMNN is compared with 2-DMNN
- Analog building blocks are identified to model the 2-DMNN based on the literature review
- Suitable modifications are made in the design of analog blocks to suite the requirements of 2-DMNN, multipliers and nonlinear transfer function
- Analog multiplier based on Gilbert cell, algorithm for neuron activation function and backpropagation algorithm is proposed, designed simulated and implemented using Cadence Virtuoso and Spectre
- Performances of 2-DMNN is analyzed and modifications are made
- Hybrid 2-DMNN architecture is designed, simulated and implemented using Cadence Virtuoso and Spectre

- Image compression and decompression is performed for different compression ratio and checked for MSE and PSNR using the test setup
- Hardware complexities are compared with existing architectures
- Schematic for the building blocks is drawn using Virtuoso Schematic Editor and simulated using Spectre
- Layout of basic blocks is drawn using Virtuoso Layout Editor and Layout validations (DRC & LVS) has been carried out for it using Assura
- Test setup is designed and implemented in Cadence Virtuoso and HSpice
- Schematic for the basic blocks is drawn using Virtuoso SE and layout is generated using Virtuoso XL. The layout is validated using Assura
- Identified architectures of neural cell is designed in SPICE deriving transistor sizes using design equations
- Designed neural cell is simulated and characterized using HSpice and the results are compared with the specifications
- Neuron cell architecture best suited for reduced leakage power and enhanced data stability is selected
- Schematics for two dimensional neural network architecture blocks are drawn in Virtuoso Schematic Editor using 0.18 µm CMOS technology.
- Layouts of all the blocks of two-dimensional neural network architecture is verified (LVS and DRC) using Assura
- Layouts for two dimensional neural network architecture is designed using Virtuoso Layout Editor
- RC extraction of the implemented two dimensional neural network architecture is performed using Assura
- Post-layout simulation for two-dimensional neural network architecture is carried out using Spectre. Simulation results are verified for reduction in leakage power and enhanced data stability
- GDSII is generated for the two dimensional neural network architecture

In the next chapter new architectures for image compression using neural network is proposed, designed, modeled and implemented. Software models for the proposed model are developed using MATLAB, test images are used to train the network. New algorithms for training are proposed. Based on the trained network different images are used to test the network performances with respect to image compression. MSE, PSNR and maximum error is calculated for standard test images. The results are compared with conventional and neural network techniques.

Chapter-4 2-D Multilayered Neural Network: Design and Implementation

4.1 Design Overview

The present research relates to neural networks for image compression, more particularly design and development of two-dimensional multilayer neural network architecture for image compression and decompression. A single chip solution which addresses computational complexities, operates faster, consumes relatively low power, and occupies less space is designed and implemented. This architecture can be scaled in a planar or massively parallel, stacked arrangement to handle more data points achieving greater processing rates. The programmability of the synaptic connections, through binary weights that are obtained during offline training makes the design reconfigurable.

Image sensors arranged in 2-D matrix of size [256 x 256], [512 x 512] and [1024 x 1024] capture light intensities that fall on them from the image. Two-dimensional multilayer neural network architecture proposed in this work captures this voltage or current values for image compression. The advantage of the proposed architecture is that it avoids image reordering (2-D to 1-D conversion as in conventional processing) and also processes the image in analog form and hence avoids analog to digital conversion. The architecture is first trained to learn the properties of images using training data sets and this makes the proposed neural network more generic and achieves better performance. Building blocks for proposed neural network architecture is identified, designed, modeled and simulated. Test setup is developed to verify functionality of the design as image compressor and decompressor. The hardware implementation of the proposed architecture is optimized for area, power and speed performances.

4.2 Design Requirements

The frame rate of a video signal captured using camera may be as slow as 30 Hz and fast as 1000 Hz. Each frame may be represented using [256 x 256] CCD arrays. Each frame of size [256 x 256] pixels with a frame rate of 30 Hz needs a processing time of 65536 pixels/33 ms. If the image is sub-divided into sub-blocks of (4 x 4) computation speed is 4096 sub-blocks/33 ms. Considering a fast motion picture at 1000 Hz the computational speed required is 4096 sub-blocks/1ms. In case of 1024 x 1024 image with a frame rate of 30 Hz requires a computational speed of 65536 sub-blocks per 33 ms and with frame rate of 1000 Hz the computational speed required is 65536 sub-blocks per 1 ms. Neural network architecture required to process images should be able to process data at this required rate. Fig. 4.1 shows the block diagram of the proposed architecture here with called as Two-Dimensional Multilayer Neural Network Architecture (TDMNN) for image compression and decompression. The architecture shown in Fig. 4.1 is implemented using VLSI technology.

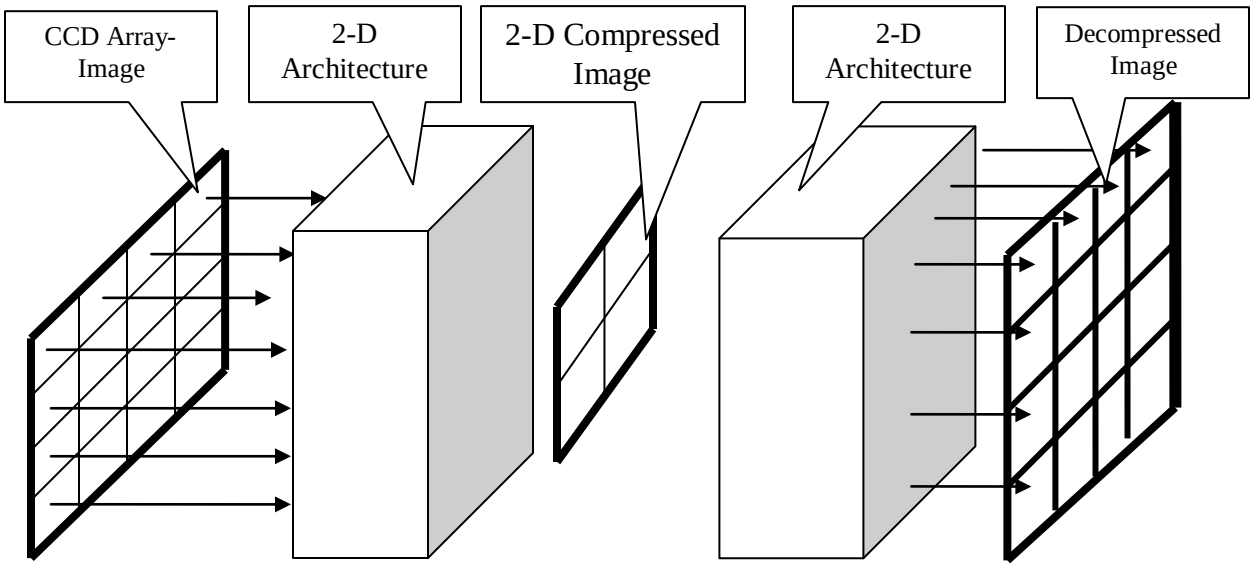


Figure 4.1 2-D Architecture for image compression and decompression

4.2.1 Two-Dimensional Multilayer Neural Network Architecture

Two-dimensional multilayer neural network architecture consists of input layer, hidden layer and output layer. The input layer consists of image pixels that are obtained from CCDs. Hidden layer consists of multiple single layer neurons arranged as shown in Fig. 4.2. Each single layer of neuron consists of multiple neurons. The output layer consists of multiple single layer neurons as shown in Fig. 4.2. In Fig. 4.2, an input image of size (8 x 8) consisting of 64 pixels is processed by a two-dimensional hidden layer consisting of four single layer neurons. In the hidden layer there are 16 neurons, each single layer neuron consisting of 4 neurons.

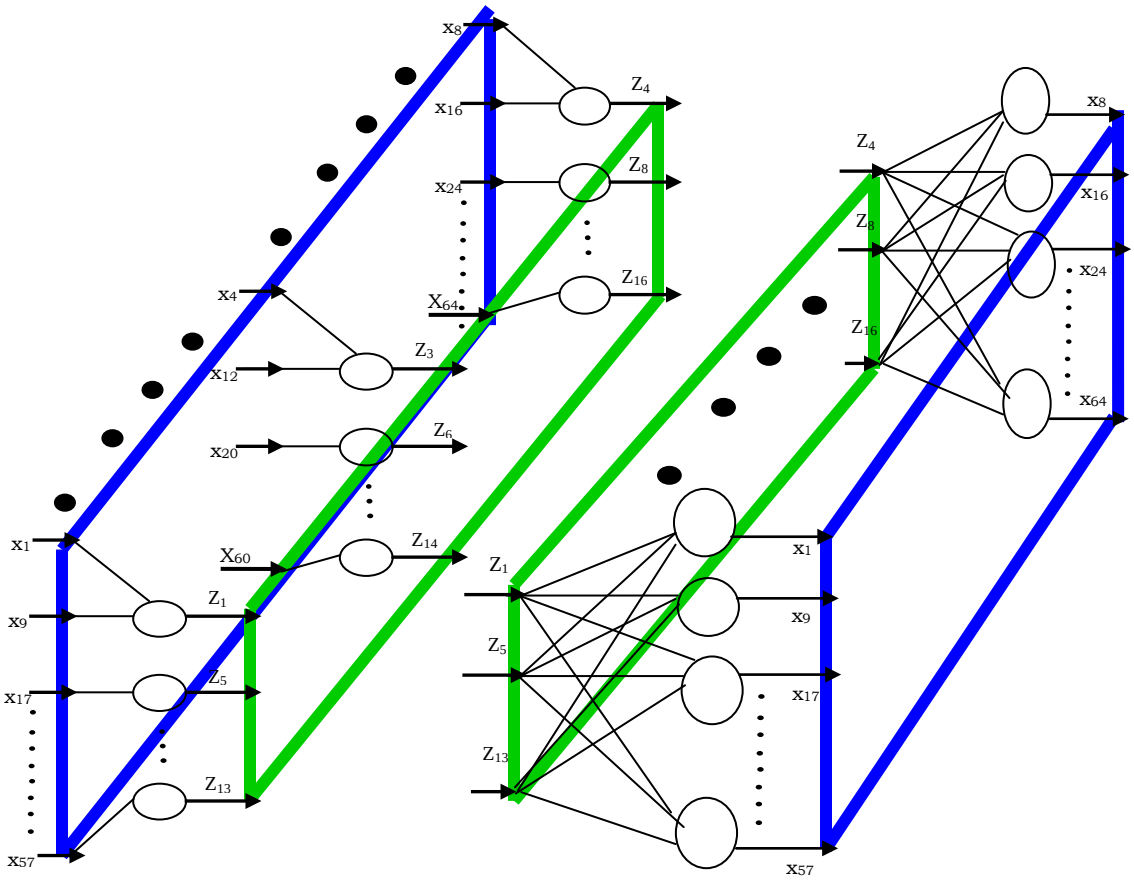


Figure 4.2 2-Dimensional multilayer neural network

As shown in Fig. 4.2, first row of inputs consists of pixels x_1 to x_8. First column of input consists of 8 pixels as shown in Fig. 4.2. 8 x 8 input image processed by the 2-D hidden layers compresses the input to 4 x 4 outputs. Compressed outputs are z_1 to z_{16} obtained in 2-D form as shown in Fig. 4.2. The output layer consisting of eight single layer neurons (each single layer neuron consists of 8 neurons), decompresses 16 compressed data to 64 outputs. The decompressed outputs are obtained in 2-D form as shown in Fig. 4.2. 2-DMNN is trained with known image data sets. Once the network is trained the network remembers the properties of different images that were used for training. The weights and bias elements of the network obtained during training are responsible in compressing and decompressing images without loss of data. Once training is accomplished, the hidden layer and the output layer are separated and are used as compressor and decompressor as shown in Fig. 4.2. The hidden layer compresses images received from image sensors, compressed data is transmitted and the output layer decompresses the received data. This research work focuses on design and development of compressor and decompressor unit as shown in Fig. 4.2.

4.2.2 Two-DMNN Parameters

Image inputs consist of [256 x 256] pixel values are sub-divided into sub-blocks and are processed by the two-dimensional network. The hidden layer compresses the sub-blocks of image; the output layer decompresses the compressed sub-blocks. For a given image the network functionalities are:

- Extracting sub-blocks from the image
- Compression of each sub-block
- Decompression of compressed sub-blocks
- Rearranging sub-blocks into image

The parameters for the 2-DMNN are as follows:

Image size: Size of input image considered for compression and decompression for example [256 x 256], [512 x 512]. In general, image can be represented as [M x M], where M is an integer.

Image sub-block size: Images are divided into sub-blocks of images of smaller size. Sub-block image size can be (16 x 16), (8 x 8), (4 x 4). In general sub-block image size can be represented as (N x N). N is an integer. Size of N is always less than M.

Hidden layer size: Size of hidden layer decides the compression ratio. Image sub-blocks are processed by the hidden layer. The number of neurons in each single layer of neuron, and the number of single layers decides compression. For example, if the sub-block image size is (8 x 8), selecting a hidden layer consisting (4 x 4) neurons achieves 75% compression. If the sub-block size is increased to (16 x 16) and is compressed using 4 x 4 hidden layers, compression of 93.75% is achieved.

Output layer size: Number of neurons in the output layer is equivalent to the sub-block image size. The compressed data at the output of hidden layer to be reconstructed to original image of size N x N, the output layer should consist of N x N neurons.

Weights and biases of network: Hidden layer and output layer consists of weights and biases that are obtained during training. Once the network is trained for various sets of images during the training phase, optimum weights and biases obtained are used to compress and decompress images. Number of weights and number of biases in the hidden layer and output layer depends upon the number of neurons and the input size. For example, if the input sub-block size is (8 x 8), and the number of hidden layer neurons is (4 x 4), the hidden layer consists of 1024 weights and 16 biases. The output layer consists of 1024 weights and 64 biases. Table 4.1 presents TDMNN parameters. For image size of (256 x 256) TDMNN network parameters are presented.

Table 4.1 TDMNN design parameters

Image size	Sub-block image size (number of input pixel)	Number of sub-blocks	Hidden layer size (i x j) (number of compressed pixels)	Out-put layer size	Compression ratio (%)	Number of weights		Number of Biases	
						Hidden layer (W_{ij})	Out-put layer (W_{ji})	Hidden layer	Out-put layer
256 x	32 x 32	0064	20 x 20 (400)	32 x	60.93	409600 (400 x 1024)	409600 (1024 x 400)	400	1024
			16 x 16		75.00	262144 (256 x 1024)	262144 (1024 x 256)	256	1024

256	(1024)		(256)	32					
			10 x 10 (100)		90.23	102400 (100 x 1024)	102400 (1024 x 100)	100	1024
			8 x 8 (64)		93.75	065536 (64 x 1024)	065536 (1024 x 64)	064	1024
			4 x 4 (16)		98.43	016384 (16 x 1024)	016384 (1024 x 16)	016	1024
			2 x 2 (4)		99.60	004096 (4 x 1024)	004096 (1024 x 4)	004	1024
			1 x 1 (1)		99.90	001024 (1 x 1024)	001024 (1024 x 1)	001	1024
256 x 256	16 x 16 (256)	0256	15 x 15 (225)	16 x 16	12.10	057600 (225 x 256)	057600 (256 x 225)	225	0256
			12 x 12 (144)		43.75	036864 (144 x 256)	036864 (256 x 144)	144	0256
			10 x 10 (100)		60.93	025600 (100 x 256)	025600 (256 x 100)	100	0256
			8 x 8 (64)		75.00	016384 (64 x 256)	016384 (256 x 64)	064	0256
			4 x 4 (16)		93.75	004096 (16 x 256)	004096 (256 x 16)	016	0256
			2 x 2 (4)		98.43	001024 (4 x 256)	001024 (256 x 4)	004	0256
			1 x 1 (1)		99.60	000256 (1 x 256)	000256 (256 x 1)	001	0256
256 x 256	8 x 8 (64)	1024	7 x 7 (49)	8 x 8	23.43	003136 (49 x 64)	003136 (64 x 49)	049	064
			6 x 6 (36)		43.75	002034 (36 x 64)	002034 (64 x 36)	036	064
			5 x 5 (25)		60.93	001600 (25 x 64)	001600 (64 x 25)	025	064
			4 x 4 (16)		75.00	001024 (16 x 64)	001024 (64 x 16)	016	064
			3 x 3 (9)		85.93	000576 (9 x 64)	000576 (64 x 9)	009	064
			2 x 2 (4)		93.75	000276 (4 x 64)	000276 (64 x 4)	004	064
			1 x 1 (1)		98.43	000064 (1 x 64)	000064 (64 x 1)	001	064
256 x 256	4 x 4 (16)	4096	5 x 3 (15)	4 x 4	06.25	000240 (15 x 16)	000240 (16 x 15)	015	016
			4 x 3 (12)		25..00	000192 (12 x 16)	000192 (16 x 12)	012	016
			4 x 2 (8)		50.00	000128 (8 x 16)	000128 (16 x 8)	008	016

			3 x 2 (6)		62.50	000096 (6 x 16)	000096 (16 x 6)	006	016
			2 x 2 (4)		75.00	000064 (4 x 16)	000064 (16 x 4)	004	016
			2 x 1 (2)		87.50	000032 (2 x 16)	000032 (16 x 2)	002	016
			1 x 1 (1)		93.75	000016 (1 x 16)	000016 (16 x 1)	001	016

In Table 4.1, a 256 x 256 image is considered as an example, and the network parameters are presented for various compression ratios. Image is sub-divided into sub-block of size (32 x 32), (16 x 16), (8 x 8) and (4 x 4). As the sub-block size is reduced the number of sub-blocks increases and hence the network requires more time to process data. As the input image is considered as 256 x 256, the sub-block size is chosen as even integer only. Hidden layer sizes presented in Table 4.1 are randomly selected only to demonstrate the network parameters. Hidden layer size is expressed as (i x j), which implies that there is j number of single layer neurons, each single layer neurons consists of i number of neurons. The value of i and j can be even or odd. Total number of neurons in the hidden layer is i multiplied by j. Number of neurons in the output layer is dependent on the sub-block image size. If the sub-block image size is 8 x 8, then the output layer consists of 8 single layer neurons, each single layer consisting of 8 neurons. Size of hidden layer decides the compression ratio. Table 4.1 also presents the number of weights required in the hidden and output layer. The network architecture is feed forward and fully connected architecture, every input reaches every hidden layer neuron. Every input pixel is correspondingly multiplied by a weight value; the multiplied inputs are added at the output of neuron. Every single neuron receiving 64 inputs are multiplied by 64 weights. If there are 16 neurons and 64 inputs than the total number of multiplications are 1024. Number of weights and number of multiplications for hidden layer and output layer are presented in Table 4.1. The arrangement of weights in the hidden layer and the output layer is highlighted in yellow. Number of biases is dependent on number of

neurons. Compression Ratio (CR) is expressed in percentage and is defined as in equation 4.1,

$$CR = (K_1 - K_2)/ K_1 \quad (4.1)$$

Where, K_1 is number of data samples in the input and K_2 is number of data samples in the compressed output.

Table 4.1 presents the compression ratios for various hidden layer sizes. Images sub-divided into sub-blocks of size (N x N) consists N^2 pixels. 2-DMNN consisting of hidden layer of size (M x M) compresses N^2 pixels to M^2 pixels. In Table 4.1 column 2 and column 3, highlighted values correspond to number of pixels in the compressed data to input data. Based on the parameters provided in Table 4.1, network parameters can be selected. In this work, two-dimensional multilayer neural network architecture with 4 x 4 input layers, 2 x 2 hidden layers and 4 x 4 output layer is chosen analyzed for its performance. Fig. 4.3 shows the compression unit of two-dimensional multilayer neural network architecture. Inputs from image sensor are represented as I denoting current (is also represented as X). Hidden layer outputs are represented as Z. Inputs that are arranged in 4 x 4 matrix form are fed into the hidden layer for compression. Every pixel in the input layer is multiplied by weight values corresponding to the neuron. As shown in Fig. 4.3, hidden layer has two layers of neuron, each layer consisting 2 neurons represented by $Z_{n,k}$. Where n represents the neuron number and k represents the network layer. For example, $Z_{1,2}$ represents first neuron in the second layer. Every neuron has 16 weight values and one bias element as per the network parameters data presented in Table. 4.1. Hence every neuron requires 16 multipliers and one adder. Output of every adder is further processed using network function.

The TDMNN architecture has 4 neurons arranged in two-dimensional structures as shown in Fig. 4.3, to each of the 4 neurons, 16 input image pixels arranged in two-dimensional format as (4 x 4) is applied in parallel. Each of the neurons produces one output which is a function of 16 input values. Input image size of (4 x 4) is processed by a (2 x 2) neuron producing (2 x 2) outputs. For simplicity and mathematical analysis, we

represent the input and the output matrix in (16 x 1) and (4 x 1) matrix form respectively. However, for hardware implementation, 2-D input is considered. Each column of (16 × 1) linear matrix of the (16 × M) matrix considered as the input layer is fed in to the two-layered network as shown in Figure 4.1. The first stage has input layer of size (4 x 4), hidden layer of size (2 x 2) (4 neurons) neurons. (4 x 4) image blocks is compressed to (2 x 2) image block at the hidden layer.

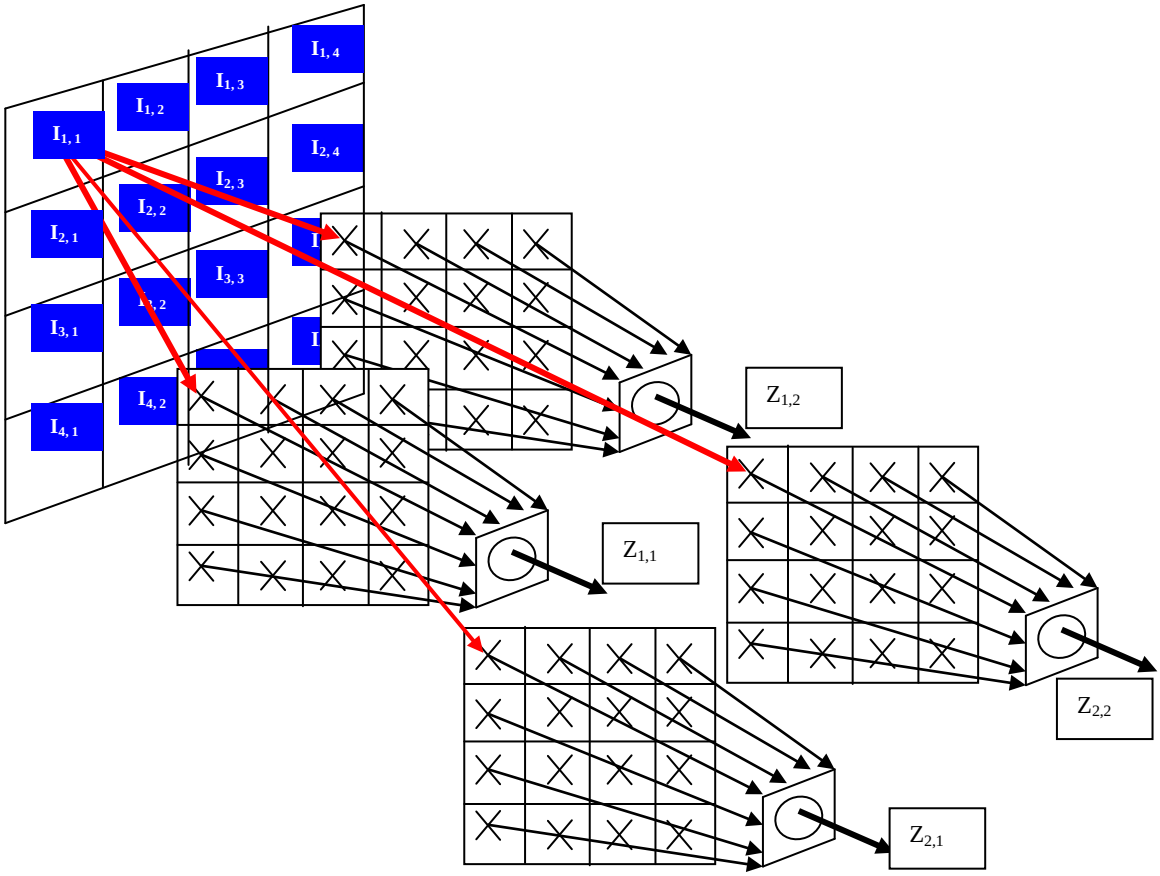
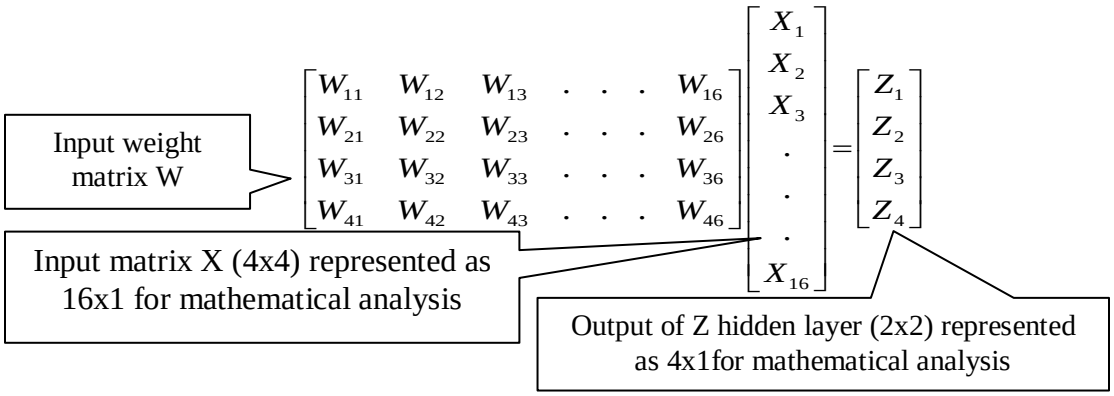


Figure 4.3 2-D neural network architecture

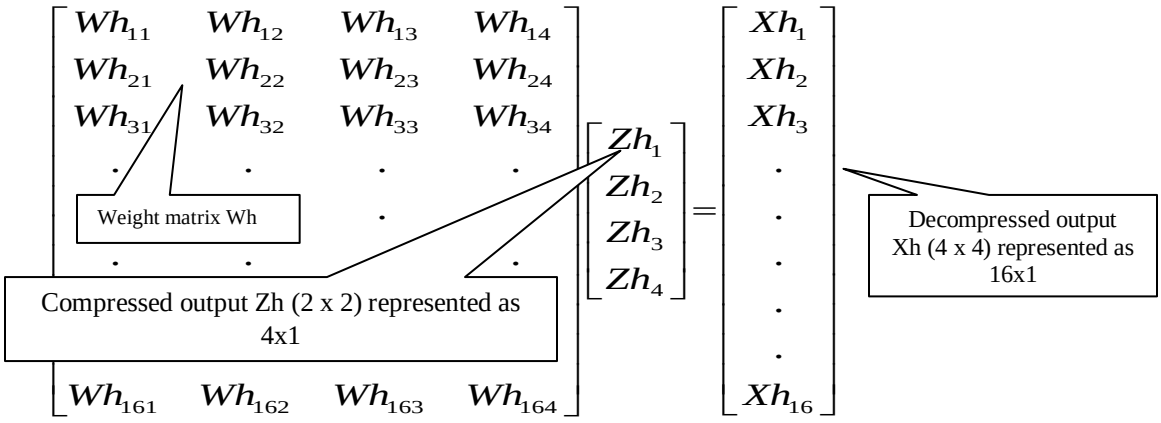
As illustrative example, let the input matrix be of size (4 x 4) arranged as (16 x 1), is compressed to (2 x 2) and arranged as (4 × 1) by the first layer as per the equation (4.1).



$$[W_{f\,x\,i}\,(4 \times 16)] * [X_{i\,x\,1}\,(16 \times 1)] = [Z_{f\,x\,1}\,(4 \times 1)] \quad (4.1)$$

The compressed (4 × 1) matrix is decompressed to (16 × 1) by the output layer as shown in equation (4.2). The output layer has 16 neurons.

$$[Wh_{i\,x\,f}(16 \times 4)]*[Zh_{f\,x\,1}(4 \times 1)] = [Xh_{i\,x\,1}\,(16 \times 1)] \quad (4.2)$$



Similarly at the receiver end, the 2 x 2 (4 x 1) matrix is reconstructed to 4 x 4 (16 x 1), by the weight matrix as shown in equation (4.2). $X_{i\,x\,1}$ is the input matrix compressed to $Z_{f\,x\,1}$ by the $W_{f\,x\,i}$ matrix at the hidden layer. $Z_{f\,x\,1}$ compressed data is transmitted, due to channel error $Zh_{f\,x\,1}$ is the received data. $Zh_{f\,x\,1}$ is decompressed to $Xh_{i\,x\,1}$ by the $Wh_{i\,x\,f}$ matrix at the output layer. $W_{f\,x\,i}$ and $Wh_{i\,x\,f}$ weight values are obtained during training of the neural network using known image data sets.

Fig. 4.4 shows the detailed view of a single neuron. 16 synaptic weights are multiplied with 16 inputs. The multiplied partial products are added as shown in equation

(4.3). At the adder stage a bias value is added and the result passed through a transfer function as shown in Fig. 4.4. As there are four neurons each produces one output and hence 4 outputs are generated from 16 inputs. Each neuron requires 16 multipliers, adders and one network function to realize equation (4.4).

$$N_1 = W_{1,1} X_1 + W_{1,2} X_2 + W_{1,2} X_3 + \text{----------} + W_{1,16} X_{16} + \text{Bias} \quad (4.3)$$

$$Z = F(N_1) \quad (4.4)$$

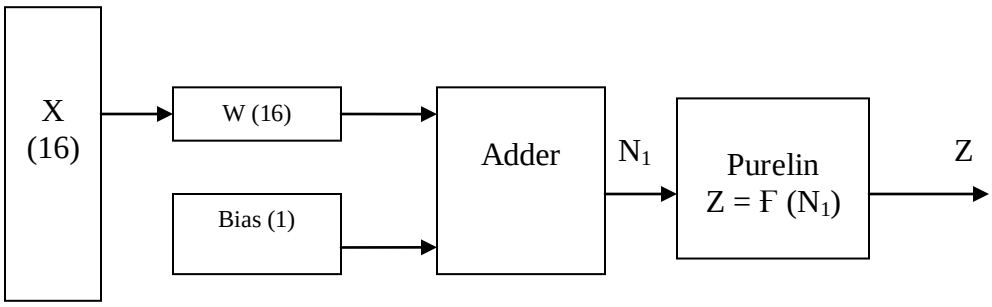


Figure 4.4 Single neuron structure

4.3 Neural Network Training

The training of the network is a very important step and need to be carefully carried out to get better performance of the network. The images shown in Fig. 4.5 are selected for the purpose of training the network.

The training sets that are used should support in obtaining an optimum weight and bias values of the network so that the image is reconstructed from the compressed data. Any image data consists of edges, vertical lines, horizontal lines, curves, diagonal lines, sharp discontinuities, intensity variation, contrast variation and random variations. Image data sets selected for training the network and presented in Fig. 4.5 have this information. Various images that have been chosen for training the network having some of the properties mentioned have been presented in Appendix – B. Training the network using these images would generalize the network and can be used for compression of any image having the above features.

These images have been removed

Figure 4.5 Image data sets selected for training the neural network

4.4 Two-Dimensional Network Training using Backpropagation

Backpropagation training algorithm is adopted for training the network. Backpropagation algorithm is used to find weights and bias elements of neural network architecture. In this technique, the error between the desired output and the network output is used to compute the new weights and bias elements. This process is carried out until the error between the target and the output is zero or within threshold. Backpropagation technique is discussed in Appendix - A works on 1-D inputs and produces 1-D output and the network is multilayered. The network proposed in this work is two-dimensional and multilayered, and the input-output is two-dimensional. Hence in this section, two dimensional multilayered neural network training algorithm based on back propagation is discussed. Fig. 4.8 shows the two-dimensional multilayered neural network architecture.

In order to train two-dimensional architecture, a two-dimensional backpropagation algorithm is required. As per the literature review carried out and summarized in chapter 2, two-dimensional training algorithm does not exist. Hence in this work two-dimensional network is trained based on modified one-dimensional backpropagation training algorithm. For multilayer neural network architecture, input image which is two-dimensional is rearranged into one-dimension as shown in Fig. 4.6. The feed-forward multilayer neural network is trained by setting the rearranged input as the target. The network is trained to reproduce the target; the training is done iteratively by updating the weights and biases based on error computed between actual output and the target. A detailed discussion on backpropagation algorithm is presented in Appendix – A.

Fig. 4.6 shows the image reordering from 2-D to 1-D. Careful observation of the reordering indicates that the adjacent pixels in every row when reordered are placed in column. In the original matrix which is two-dimensional pixel 1 is adjacent to pixel 2, pixel 2 is adjacent to pixel 3, and so on. In other words, neighboring pixels are adjacent to each other. When the two-dimensional matrix is rearranged into one-dimension matrix as shown in Fig. 4.6, pixel 5 that was adjacent to pixel 1 is actually moved down in this column matrix. In an image neighboring pixels are highly correlated. Due to reordering the correlation between pixels is lost. Hence the correlation between the pixels in multiple rows is lost and hence the 1-D training algorithm cannot be used to train the proposed two-dimensional network. In order to extend the one-dimensional training algorithm to the proposed two-dimensional network, image reordering process is modified and the training sets are created to maintain the pixel-pixel correlation. Retaining the correlation between neighboring pixels in a given image, existing one dimensional backpropagation training technique is used to train the two-dimensional network. In Fig. 4.7, modified reordering scheme (Fisher, Perkins, Walker and Wolfart 2003) is demonstrated to increase the correlation between pixels. In this scheme, pixels 1, 2, 5 and 6 that are very close are reordered into column matrix. By doing so the pixels are

placed closer than the previous technique. Further, the correlations between pixels are further retained by using overlapping sub-blocks as shown in Fig. 4.7.

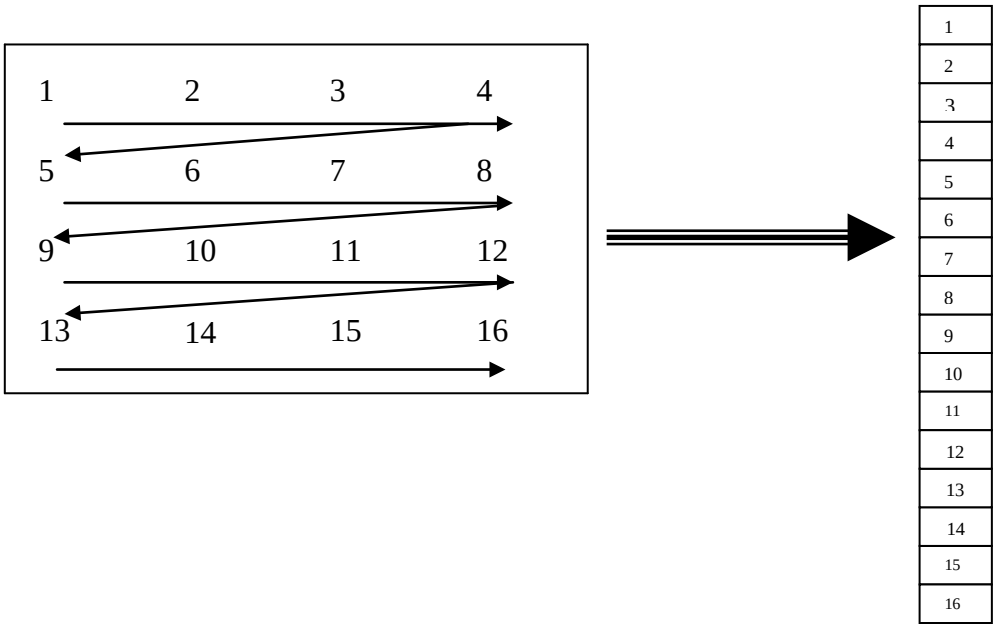


Figure 4.6 Image reordering from 2-D to 1-D

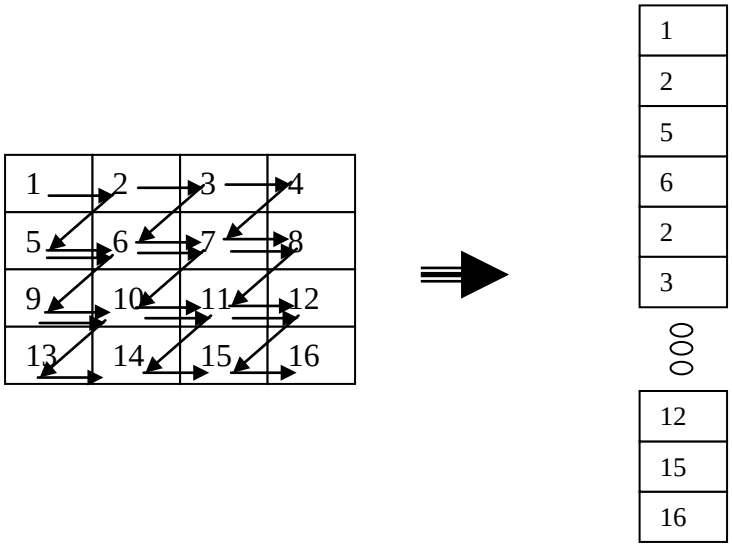


Figure 4.7 Modified reordering scheme to improve correlation between pixels

Comparing the reordered outputs presented in Fig. 4.5 and Fig. 4.7, in Fig. 4.7 adjacent pixels in a sub-block are placed closer to each other and hence correlation

between pixels is retained to an extent. Training the two-dimensional network using backpropagation algorithm based on this reordered matrix optimum weights and biases are obtained for the network.

Since the reordering of the matrix ensures that the 2-D pixel-pixel correlation is not affected by the modified reordering scheme. Backpropagation algorithm for feed-forward multilayer network can be extended to 2-D multilayer network. Before training the 2-D network the reordering of the matrix as shown in Fig. 4.7 is required. A software reference model is developed for the proposed two-dimension multilayer neural network is designed, modeled, simulated and results are verified for multiple test cases. The network model is trained using the reordered data set, and simulated using different image samples to validate its performances. Weights and biases obtained after training is used to compress and decompress the network. In Fig. 4.8, the compressor unit is presented.

As shown in Fig. 4.8, for the compressor unit consisting of 4 neurons in the hidden layer arranged as 2 x 2 is fed with 16 inputs that are available as 4 x 4. The input image pixels $X_1 – X_{16}$ are the pixels of a sub-image arranged as 4 x 4 matrixes. Every neuron in the hidden layer receives these inputs but is correspondingly multiplied by the weights. First neuron weight matrix $W_{1,1} – W_{1,16}$ is multiplied by input image $X_1 – X_{16,}$ second neuron weight matrix $W_{2,1} – W_{2,16}$ is multiplied by $X_1 – X_{16,}$ similarly $W_{3,1} – W_{3,16}$ is multiplied by $X_1 – X_{16}$ and $W_{4,1} – W_{4,16}$ is multiplied by $X_1 – X_{16.}$ The partial products are added and processed using the network function and Z_1- Z_4 the compressed data is computed. During the training process the image pixels $X_1 – X_{16}$ are arranged in one-dimensional matrix form. During implementation the input image pixels $X_1 – X_{16}$ are arranged in 4 x 4 matrix forms. Hence modified one-dimensional training algorithm is adopted for two-dimensional networks. Fig. 4.8 shows the input image sub-block consisting of 4 x 4 pixels arranged in X-Y plane is compressed to 2 x 2 pixels represented in X-Y plane.

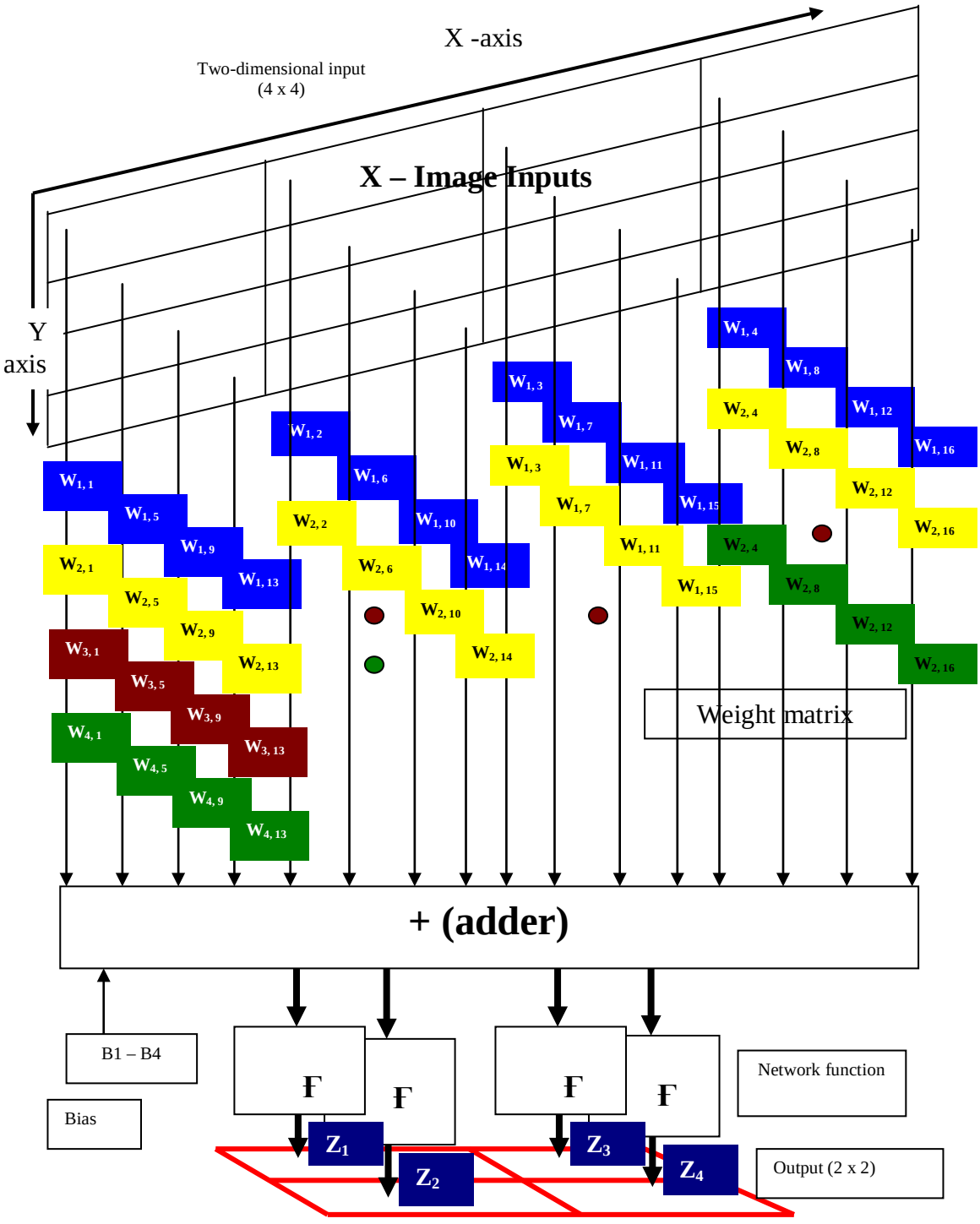


Figure 4.8 Two-dimensional neural network architecture (Compressor unit)

As shown in Fig. 4.8, sub-block image of size 4 x 4 is compressed to 2 x 2 block. The hidden layer has 4 neurons, 16 inputs to each neuron is multiplied by 16 weight values. The multiplied partial products are added and processed by the network function. 2 x 2 neurons produce 4 outputs that are compressed image data set for an input image of size 4 x 4 sub-blocks. To compress each sub-block, multiplication, addition and network function are the three major processing blocks required. Network functions used classify the network as linear neural network and nonlinear network. Hardware realization of the proposed architecture in VLSI requires designing multipliers, adders and network function. Weight and biases need to be stored using on-chip memory. Training of the network is required to find optimum weights and biases for the network. Test inputs are required to validate the network performances.

4.5 Design, Modeling and Analysis of TDMNN Architecture

In this section, design and analysis of two-dimensional multilayer neural network architecture is presented. In this section, for analyzing the performances of the network software reference model is designed. Fig. 4.9 shows the top level architecture of the software reference model of the proposed architecture. N x 1 pixel is processed to K x 1 pixel by the compression block. The compressed output is reconstructed to N x 1 pixels by the decompressor unit as shown in Fig. 4.9.

The basic building blocks for the network are:

1. Weight matrix (required in hidden layer and output layer)
2. Bias (required in hidden layer and output layer)
3. Multipliers and Adders
4. Network function (required in hidden layer and output layer)

In this work, the following network functions have been used for the analysis of the network:

1. Purelin or purely linear function
2. Tansig

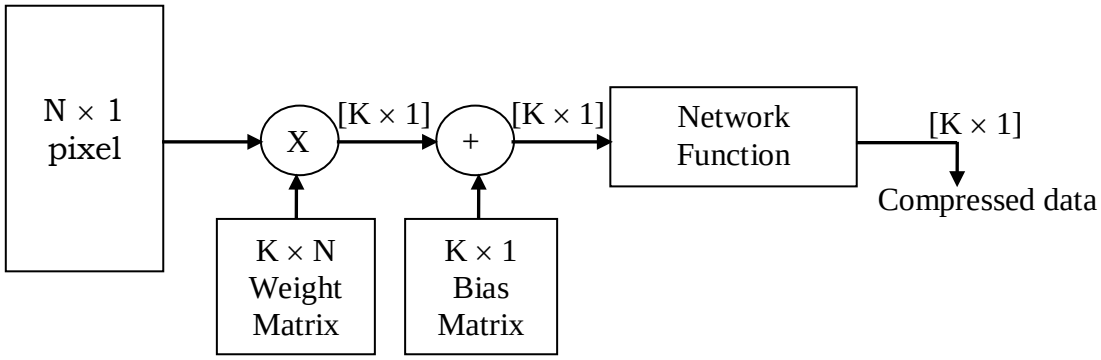


Figure 4.9(a): Compression block diagram

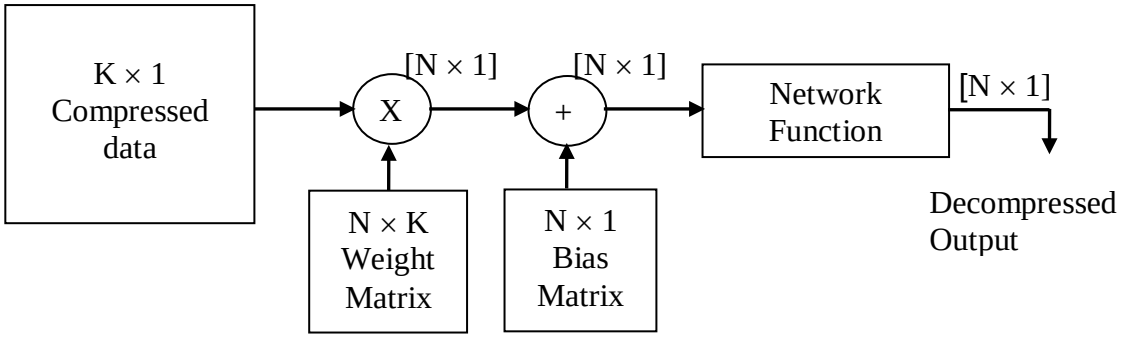


Figure 4.9(b): Decompression block diagram

Figure 4.9 level block diagram of neural network architecture for software model

Fig. 4.10 shows the input-output relation for the selected network functions. In a Purelin function, there is a linear relationship between input and output. In Tansig and Logsig functions input-output relationship is nonlinear. Network functions shown in Fig. 4.10 are standard functions used to design neural network architectures. Based on the kind of network function used, the neural network architecture can be classified as linear, non-linear and hybrid network. Table 4.2 shows the types of networks and their classification. The hidden layer has 4 neurons and output layer has 16 neurons, hence 4 network functions are required in the hidden layer and 16 network functions are required in the output layer. Choosing appropriate network functions for a given application is a very

important step. As per the literature review carried and presented in Chapter 2, linear and nonlinear networks are used for image compression.

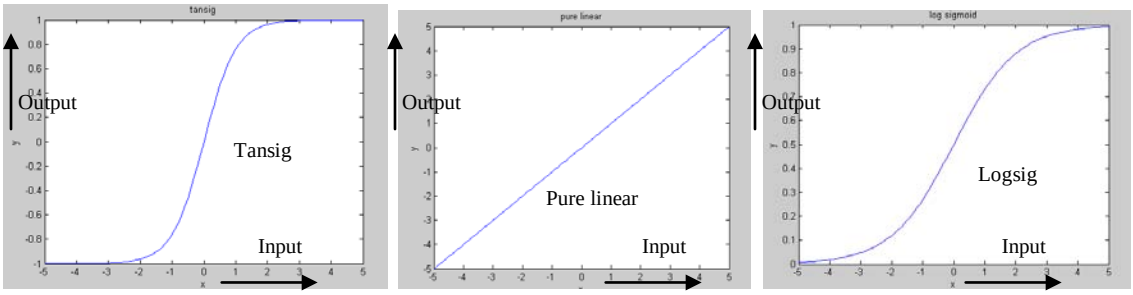


Figure 4.10 Network functions Tansig, Pure linear and Logsig

In this research work, network performance analysis is carried out using the network functions shown in Table 4.2. A hybrid network is in which, the hidden layer consists of nonlinear network function (tansig) and output layer consists of linear network function (purelin).

Table 4.2 Neural network classification based on transfer function

Neural network type	Network function	
	Hidden layer	Output layer
Linear network	Purelin	Purelin
Nonlinear network	Tansig or logsig	Tansig or logsig
Hybrid network	Tansig or logsig	Purelin

In this work, performance analysis of all the three types of network is carried out. Nine images have been selected for analyzing the performance of the network, the training images are shown in Fig. 4.5. Performance metrics such as Mean Square Error (MSE), Peak Signal to Noise Ratio (PSNR), Maximum Error and network delay (for compression and decompression) are computed for all the nine images. Variation in performance metrics for different compression ratio and bits per pixel is obtained and analyzed for all the nine images. The performance of the three networks is carried out and

compared with JPEG standard compression technique. Based on the results obtained suitable conclusions are drawn for hardware implementation.

Image quality metrics used for comparison are defined as follows:

⇒ **Image quality metrics:** Two of the image quality metrics used to compare the various image compression techniques are the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR) (Huynh-Thu and Ghanbari 2008, Thomos, Boulgouris and Strintzis 1998 and Xiangjun and Jianfei 2007).

The MSE is the cumulative squared error between the compressed and the original image, whereas PSNR is a measure of the peak error. The mathematical formulae for the two are given in equation (4.5), (4.6) and (4.7),

$$mse = \frac{1}{Q}\sum_{k=1}^{Q} e(k)^2 = \frac{1}{Q}\sum_{k=1}^{Q} (t(k) - a(k))^2 \tag{4.5}$$

Equation (4.5) is a generic equation for mean square error, where e is the error between two parameters t and a. Normalization factor Q represents the length of the sequence and k is a variable.

For image,

$$MSE = \frac{\sum_{M,N} [I_1(m,n) - I_2(m,n)]^2}{M * N} \tag{4.6}$$

Where, $I_1(m,n)$ is the original image, and $I_2(m,n)$ is the decompressed image. M and N represent the image size.

$$PSNR = 10\log_{10}\left(\frac{R^2}{MSE}\right) \tag{4.7}$$

Where, R is the maximum fluctuation in the input image data type. For example, if the input image has a double-precision floating-point data type, then R is 1. If it has an 8-bit unsigned integer data type, R is 255.

A lower value of MSE means higher value of PSNR. Lower MSE means that the decompressed image is same as the original image with minimum loss. Higher PSNR is good for better image quality. Higher value of PSNR is good because it means that the ratio of Signal to Noise is higher. Result analysis of the proposed work is based on these quality metrics. Maximum error is another quality parameter that is also considered for analyzing the performance of the network. This parameter determines the maximum error between the original and the decompressed image. The error between the original input signal and the decompressed signal is computed, and the absolute maximum error is computed, which helps in identifying the maximum deviation between the original and the decompressed image. Another factor that is also considered in for comparison is the total time required to compress and decompress the image using the proposed neural network architecture. As the network has to perform preprocessing operations on the input image, compression and decompression using weights and biases, total delay would help in estimating the complexity of the network. The quality metrics for variation in compression ratio and bits per pixel are computed for analysis.

Bits per pixel (bpp) is calculated as follows: the input image is gray scale image and each pixel is represented by 8-bit and hence the input image requires 8 bits per pixel for representation. The compressed data is also represented using 8 bits. For example, if we consider a 256 x 256 image each pixel represented by 8 bits, the total number of bits required to represent the original image is 524288, the number of bits per pixel is 8 (524288/256*256) for the original image. The original image is sub-divided into 4 x 4 sub-block, for 256 x 256 images there are 4096 sub-blocks. Each of these sub-blocks is compressed to 2 x 2 block. Hence the compressed image has 4096 compressed sub-blocks of 2 x 2 sizes. Hence there would be 16384 pixels in the compressed data. If each

pixel is represented by 8 bits the total number of bits is 131072 bits. 131072 bits are required to represent 256*256 pixels in the compressed form. Hence the number of bits per pixel is 2 (131072/256*256).

Compression Ratio (CR) is calculated as follows: each sub-block of the 256 x 256 image consisting of 16 pixels (4 x 4) is compressed to 2 pixels (for example), the compression ratio is given by: [(*number of input pixels - number of compressed pixels*)/ *number of input pixels*] %.

4.6 Design of Software Reference Model

Design and development of software reference model using Matlab consists of four major aspects:

1> Preprocessing:
   a. Selection of training image data sets
   b. Reordering of the image data sets suitable for 2D network

2> Network training:
   a. Data sets arrangement for training the network
   b. Setting of training goals
   c. Setting of number of iterations

3> Compression and decompression
   a. Compression of image using the trained network
   b. Decompression of image using the trained network

4> Performance analysis
   a. Visual analysis
   b. Mean square error
   c. Peak signal to noise ratio
   d. Maximum error
   e. Computation time

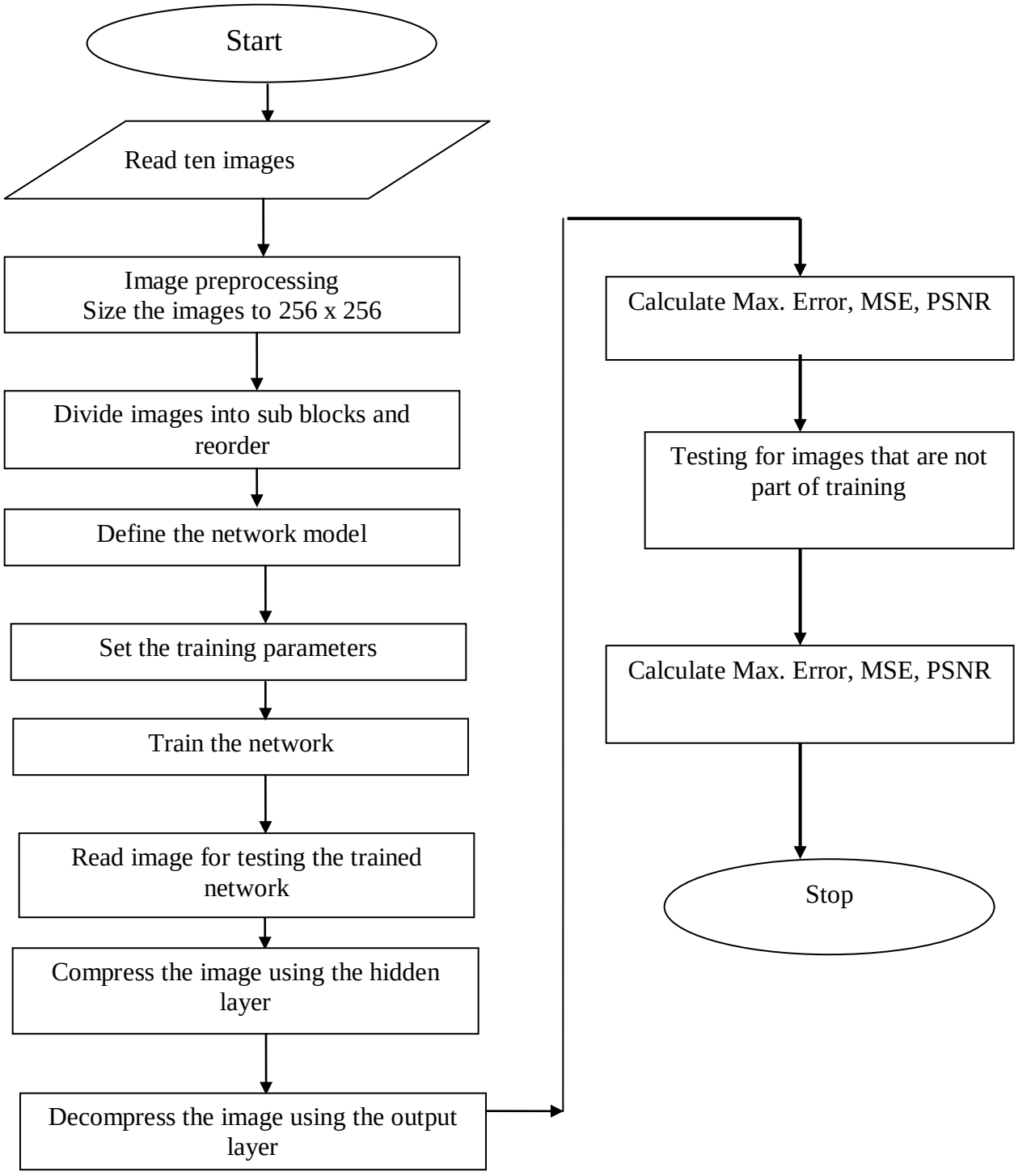


Figure 4.11 Software reference model flow chart

The flow chart for the reference model is shown in Fig. 4.11. Training data sets are selected, rearranged into one-dimensional and network architecture is modeled. Training parameters for the proposed network is set; the network is trained based on back propagation algorithm. The trained network is tested using new set of image data sets. The quality metrics are measured to analyze the network performance.

4.7 Results and Analysis

This section discusses software simulation results obtained based on the proposed neural network architecture. Gray scale images have been considered for analyzing the neural network performances. Color images can also be compressed based on the proposed two-dimensional network. The proposed method is implemented in MATLAB (R2006b) Version 7.3. Software reference model developed is presented in Appendix – A. The image is transformed, quantized, compressed using neural network and then reconstructed. In order to analyze the performances of the proposed network, quality metrics such as MSE, PSNR are considered for variations in bpp. The network is trained using the selected nine images, further to test the generalization property of the network, images that were not part of training data sets is used to analyze the network performances. For the two-dimension multilayer neural network architecture model the following are the results obtained and discussed:

1> Linear network:
   i. Quality metrics vs. bpp
2> Hybrid network:
   i. Quality metrics vs. bpp
   ii. Quality metrics vs. bpp in graph form for three images
3> Comparison of linear, nonlinear and hybrid networks
   i. Quality metrics vs. bpp
4> Comparison of hybrid, linear with JPEG technique
   i. Quality metrics vs. bpp

ii. Quality metrics vs. bpp in graph form for three images
iii. Computation time for three images

5> Noise analysis
i. Quality metrics vs. bpp is presented
ii. Visual representation of reconstructed images

6> Error analysis
i. Quality metrics vs. bpp is presented
ii. Visual representation of reconstructed images

Results for three neural networks are discussed in detail. Based on the results and observations new techniques are presented at the end of this section. Images that are not part of training data sets are also used to identify network performances. This helps in generalizing the network based on the optimum weights and bias obtained during training. Various images that have been used for validating the neural network performances are presented in Appendix-B.

4.7.1 Linear Network for Compression and Decompression

Fig. 4.12 shows the linear network architecture model, the input image is subdivide into 4 x 4 sub-blocks and reordered into 16 x 1 inputs. The input is set as the target for training purpose. The hidden network and output network consists of Purelin function and hence the network is linear.

Table 4.3 presents the quality metrics for nine different images. These images have been randomly selected. For different bits per pixel values MaxError, MSE and PSNR are computed using the neural network architecture and the results obtained are given in Table 4.3.

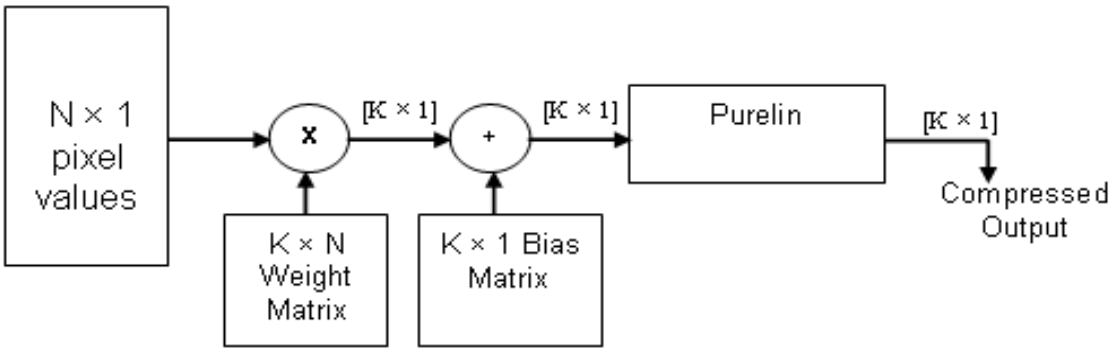


Figure 4.12 (a): Linear network for compression

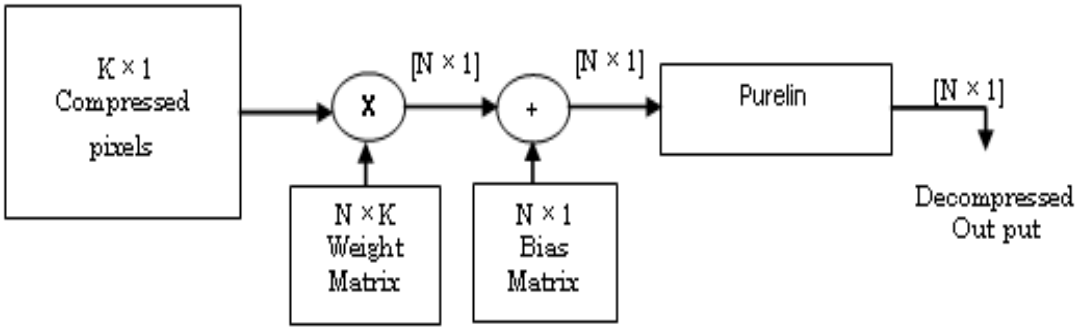


Figure 4.12 (b) Linear network for decompression

Figure 4.12 Linear neural network for compression and decompression

Table 4.3 Compression ratio vs. Quality metrics for linear network

Quality Metrics	Bits per pixel				
	0.5	2	4	5	7.5
Baboon					
MaxError	12.0	10.0	15.0	23.0	24.0
MSE	1.0	1.4	2.9	4.4	6.3
PSNR	47.7	46.6	43.4	41.6	40.1
Image1					
MaxError	15.0	23.0	33.0	27.0	28.0
MSE	2.4	5.6	7.2	6.5	8.8
PSNR	44.3	40.6	39.5	39.9	38.6
Peppers					
MaxError	55.0	61.0	99.0	74.0	68.0
MSE	15.5	24.8	25.6	28.4	36.5

PSNR	36.2	34.1	34.0	33.5	32.4
Testim					
MaxError	31.0	33.0	66.0	83.0	56.0
MSE	10.1	12.1	43.0	43.0	44.2
PSNR	38.0	37.2	31.7	31.7	31.6
Trees					
MaxError	44.0	58.0	77.0	54.0	61.0
MSE	17.0	29.5	49.7	32.3	20.9
PSNR	35.8	33.4	31.1	33.0	34.9
Pears					
MaxError	37.0	31.0	68.0	43.0	78.0
MSE	12.1	06.0	41.8	17.3	33.9
PSNR	37.2	40.3	31.9	35.7	32.8
Saturn					
MaxError	28	61	57	50	42
MSE	2.6	10.3	6.1	10.0	8.6
PSNR	43.9	37.9	40.2	38.1	38.7
Kids					
MaxError	61	42	57	60	45
MSE	19.2	15.3	20.3	20.0	17.2
PSNR	35.2	36.2	35.0	35.1	35.7
Rice					
MaxError	19	32	29	32	32
MSE	8.5	17.8	22.0	21.8	24.6
PSNR	38.7	35.6	4.7	34.7	34.2

From the results obtained it is observed that MSE is not varying uniformly for variations in bpp from 0.5 to 7.5 for all the images. Fig. 4.13 shows the plot of MSE vs. bpp for selected images from Table 4.2. From the Fig. 4.13, it is observed that MSE uniformly varies for baboon and image1, for pears and kids images MSE has random variations. In case of Trees image, MSE increases to 50 at 4 bpp and falls back to 20 at 7.5 bpp. This is due to the fact that, the neural network architecture is trained to compress

and decompress all the images based on optimum weight and biases obtained during training. Hence the weight and bias values need to be optimized for all possible images.

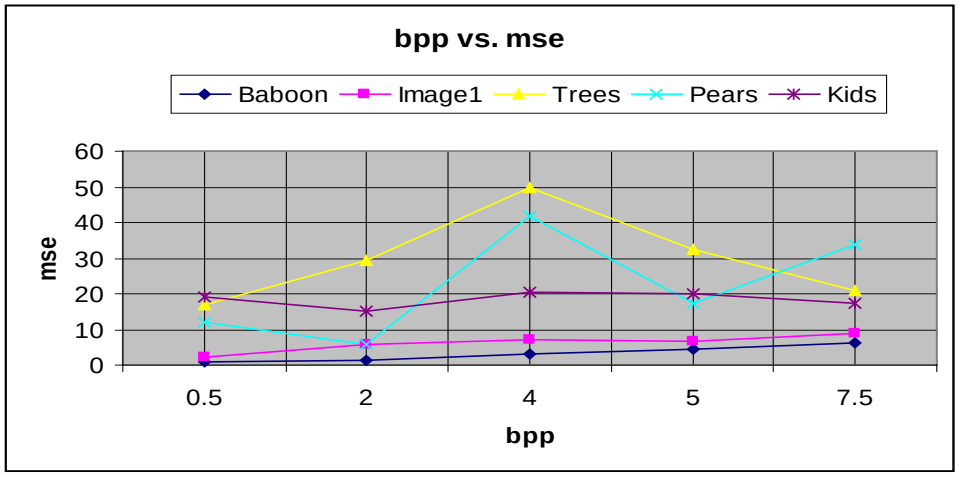


Figure 4.13 Bpp vs. quality metrics for linear network for selected images

From the results presented in Fig. 4.13 the following are the observations made:

1> For images baboon and trees, at 0.5 bpp and 2 bpp MSE is minimum, which implies that due to compression information present in the images are not lost.

2> For pears and kids images, at 2 bpp MSE is minimum, which implies the linear network, is able to reconstruct the original image at 2 bpp better than any other bpp values.

3> For trees image MSE is lowest at 0.5 bpp.

From the results obtained, the following conclusions are made:

1> Linear network reconstructs images like baboon and image1 at 2 bpp compression minimum MSE is obtained.

2> In baboon and image1, pixel-pixel correlation is very high (visual observation) and hence the two-dimension training algorithm exploits this property. In other images there are multiple edges and hence linear network performance varies.

3> Highly correlated images when sub-divided into smaller sub-blocks would not loose information. Images consisting of multiple edges when sub-divided into

smaller sub-blocks and compressed and decompressed separately using the TDMNN, when merged into original image size lose information due to blocking artifacts. Images with multiple edges to obtain better MSE, larger sub-block sizes are required.

4> Linear network performances are not same for all the images at different values of bpp. This implies that the network performance depends on input data.

In the next section, performances of hybrid neural network are discussed.

4.7.2 Hybrid Network for Compression and Decompression

Fig. 4.14 shows the hybrid network architecture model, the hidden network and output network consists of Tansig function and Purelin function respectively and hence the network is hybrid.

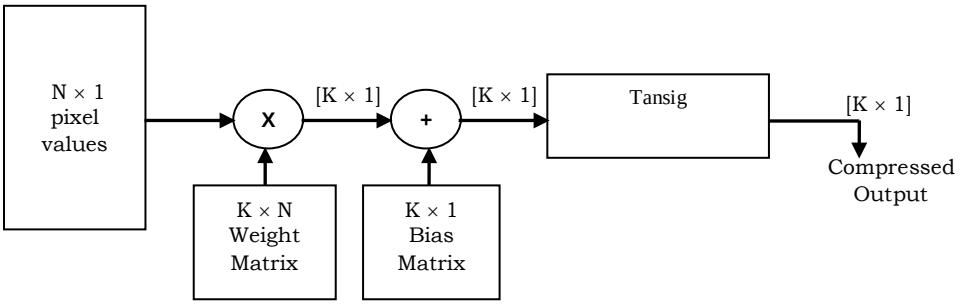


Figure 4.14(a): Compression block diagram

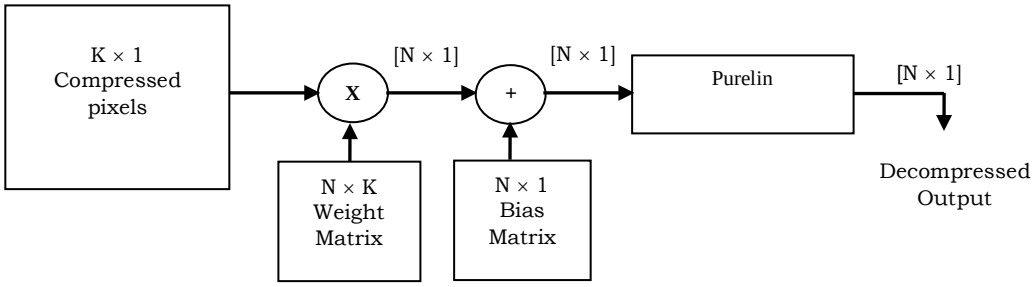


Figure 4.14(b) : Decompression block diagram

Figure 4.14 Neural network for compression and decompression

Input image is compressed using nonlinear network (hidden layer) consisting Tansig network function. Compressed data is reconstructed using linear network (output layer). In this architecture, Tansig and Purelin functions have been used in the hidden layer and output layer respectively. Logsig with Purelin, Purelin with Logsig and Purelin with Tansig function have also been used to analyze the network performance (results have not been presented). The hybrid TDMNN is trained using the modified backpropagation algorithm. Network is trained using nine test images for 100 epochs. The optimum weights and biases obtained are used to design the network. Performances of the network are obtained using multiple images and the results are presented in Table 4.4. Similar to the analysis carried out for linear network, Table 4.4 presents the quality metrics for hybrid network.

Table 4.4 Compression ratio vs. Quality metrics for hybrid network

Quality Metrics	Bits per pixel				
	0.5	2	4	5	7.5
Baboon					
MaxError	5.00	8.00	12.00	45.00	46.00
MSE	0.06	0.10	2.50	4.70	4.70
PSNR	59.90	57.80	44.10	41.30	41.40
Image1					
MaxError	3.00	4.00	47.00	48.00	64.00
MSE	0.15	00.42	15.72	18.42	19.80
PSNR	56.20	51.80	36.16	35.40	35.15
Peppers					
MaxError	2.00	91.00	96.00	133.00	140.00
MSE	0.004	55.20	48.70	98.80	76.30
PSNR	72.00	30.70	31.20	28.10	29.30
Testim					
MaxError	30.00	85.00	76.00	63.00	100.00
MSE	6.10	68.70	34.20	26.30	60.30
PSNR	40.23	29.70	32.70	33.90	30.30
Trees					
MaxError	6.00	70.00	20.00	76.00	97.00
MSE	0.01	39.48	1.20	57.23	68.40

PSNR	66.50	32.10	47.30	30.50	29.70
Pears					
MaxError	1	109	96	109	106
MSE	9.46E-04	57.05	58.19	67.17	67.18
PSNR	78.30	30.50	30.40	29.85	29.82
Saturn					
MaxError	15	37	50	68	75
MSE	1.47	4.76	16.77	22.02	24.26
PSNR	46.40	41.30	35.80	34.70	34.20
Kids					
MaxError	22.00	29.00	83.00	45.00	73.00
MSE	3.02	9.96	49.80	22.95	45.20
PSNR	43.30	38.14	31.15	34.52	31.57
Rice					
MaxError	2.00	8.00	61.00	60.00	69.00
MSE	0.08	0.96	73.70	89.31	95.40
PSNR	58.76	48.27	29.45	28.62	28.33

Fig. 4.15 compares the quality metrics of hybrid network for three different images.

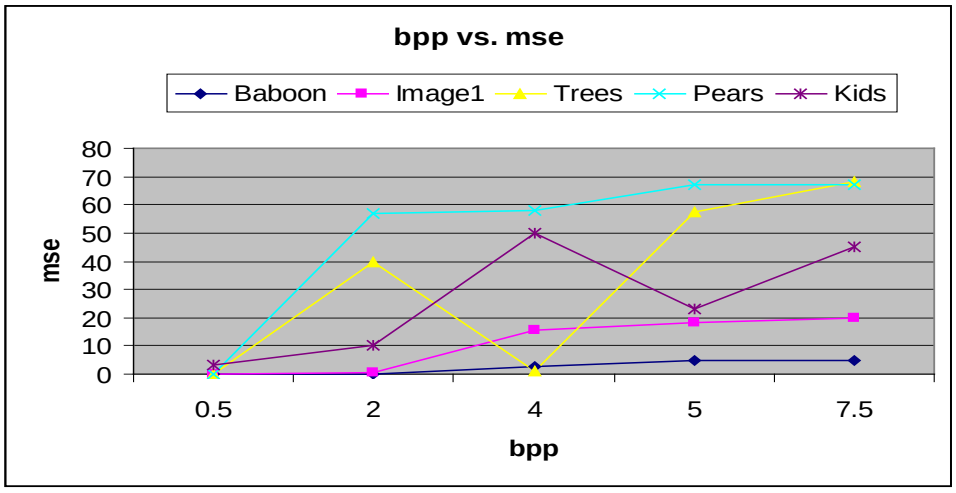


Figure 4.15 Bpp vs. Quality metrics for hybrid network for selected images

It is found that PSNR and MSE are inversely proportional. For lower bpp MSE is found to be very low and PSNR is very good. The maximum value of PSNR is 78 for

pears image at 0.5 bpp, which is better than linear PSNR for the same image and bpp. The minimum MSE is 0.009 for pears at 0.5 bpp. Fig. 4.15 presents the bpp vs. MSE for selected images such as baboon, image1, Trees, pears and Kids. It is found that MSE for Baboon and Image1 images vary similar to the linear network. In case of Trees, at 4 bpp hybrid network achieves 1.2 MSE, but with linear network MSE is 49.7. This implies that the network performance is image dependent.

From the results obtained it is found that hybrid network achieves better quality metrics compared with linear network at lower bpp. Another very interesting observation is that at higher bpp linear network perform better than hybrid networks. Based on the experimental analysis carried out the following are the observations made:

1. MSE and PSNR values obtained at 0.5 bpp for both linear and hybrid network is better than MSE and PSNR values at 7.5 bpp for all images. At 0.5 bpp, 16 pixels are compressed to 1 pixel. Hence the network consists of 16 input layers, one hidden layer and 16 output layers. At 7.5 bpp, 16 pixels are compressed to 15 pixels. This network consists of 16 input layers, 15 hidden layers and 16 output layers.
2. 16 inputs are passed into one neuron in the hidden layer. The correlation or redundancies among all the neighboring pixels are exploited and hence the network reproduces the image with less MSE and higher PSNR. At 7.5 bpp, 15 neurons are used to compress 16 pixels, and hence the correlation properties among the pixel values are not exploited. This achieves poor PSNR and MSE.

From the results obtained, the network performance is image dependent. Also training the network is a very important phase. Optimum weights and biases obtained define the network performance. Performance of nonlinear network is not presented. Comparison of network performances of all three networks is presented in next section. Table 4.5, compares the results of hybrid and linear network for the five selected images with variation in bpp. MSE for selected five images are presented with variation in bpp.

Table 4.5 Comparison of Hybrid and Linear Network

Quality Metrics	Bits per pixel				
	0.5	2	4	5	7.5
Baboon					
Linear - MSE	1.00	1.40	2.90	4.40	6.30
Hybrid - MSE	0.06	0.10	2.50	4.70	4.70
Image1					
Linear - MSE	2.40	5.60	7.20	6.50	8.80
Hybrid - MSE	0.15	0.42	15.72	18.42	19.80
Trees					
Linear - MSE	17.00	29.50	49.70	32.30	20.90
Hybrid - MSE	0.01	39.48	1.20	57.23	68.40
Pears					
Linear - MSE	12.10	6.00	41.80	17.30	33.90
Hybrid - MSE	9.46E-04	57.05	58.19	67.17	67.18
Kids					
Linear - MSE	19.20	15.30	20.30	20.00	17.20
Hybrid - MSE	3.02	9.96	49.80	22.95	45.20

From the results obtained and presented in Table 4.5, hybrid network performs better than linear network at 0.5 bpp. At higher values of bpp, linear network performs better than hybrid network. At 4 bpp for trees image hybrid achieves very less MSE compared with linear network. Form the results obtained it is concluded that the network performance is a function of input. Next section compares the performance of all three networks.

4.7.3 Performance Comparison of all Three Network Architectures

Quality metrics for all the three networks are presented in Table 4.6, the network performances have been computed for 4 bpp. From the results obtained it is found that the nonlinear network has very poor quality metrics compared with linear and hybrid network and hence only linear and hybrid network is used for image compression and decompression in this work.

Table 4.6 Quality metric for all three networks

	Nonlinear	Linear	Hybrid
Baboon			
MaxError	92.00	15.00	12.00
MSE	192.50	2.94	2.50
PSNR	25.28	43.44	44.14
Image1			
MaxError	178.00	33.00	47.00
MSE	123.47	7.28	15.72
PSNR	17.21	39.50	36.16
Peppers			
MaxError	161.00	99.00	96.00
MSE	127.44	25.63	48.73
PSNR	27.07	34.04	31.25
Testim			
MaxError	103.00	66.00	76.00
MSE	121.39	43.06	34.22
PSNR	27.28	31.78	32.78
Trees			
MaxError	193.00	77.00	20.00
MSE	310.79	49.75	01.20
PSNR	23.20	31.16	47.31
Pears			
MaxError	172.00	68.00	96.00
MSE	151.57	41.86	58.19
PSNR	26.32	31.91	30.48
Saturn			
MaxError	82.00	57.00	50.00
MSE	45.67	6.16	16.77
PSNR	31.53	40.23	35.88
Kids			
MaxError	201.00	57.00	83.00
MSE	401.50	20.32	49.80
PSNR	22.09	35.04	31.15
Rice			
MaxError	96.00	29.00	61.00
MSE	227.34	22.03	73.70
PSNR	24.50	34.70	29.45

From the results obtained the following are the observations made:

1. MSE and PSNR for all images are not the same. This shows that images don't have common information and the network has learnt to reproduce the images based on information content
2. PSNR is best for images like trees, testim and baboon using Hybrid network, for all other images linear network achieves best PSNR.

Form Table 4.6, it is found that linear and hybrid networks achieve higher PSNR and lower MSE compared with nonlinear network performances. Also it is found that, hybrid network outperforms linear network for certain images. Hence, it is concluded that hybrid and linear networks can be used depending upon the image used for compression.

Figure 4.16 Decompressed output using linear network

In order to understand the performances of all the three networks, decompressed images using all the three networks for four different images are presented in Fig. 4.16 to

Fig. 4.18. From the visual images presented in Fig. 4.16 to Fig. 4.18, it is very clear that, linear network and hybrid network have better results compared to nonlinear network. Fig. 4.16 presents the result of linear network, form the images we find that linear network has reproduced the images with minimum distortions.

Fig. 4.17 presents the decompressed images using nonlinear network. All the four images have distortions compared to the decompressed image using linear network. From the results shown in Fig. 4.18, the decompressed results using hybrid network have better visual quality compared to nonlinear network, but not better than linear networks. The same results were also observed and discussed using graphs and tables presented.

Figure 4.17 (a) Figure 4.17 (b)

Figure 4.17 (c) Figure 4.17 (d)

Figure 4.17 Decompressed output using nonlinear network (a) baboon (b) Peppers

Figure 4.18 (a) Figure 4.18 (b)

These images have been removed

Figure 4.18 (c) Figure 4.18 (d)

Figure 4.18 Decompressed output using hybrid network (a) baboon (b) peppers

4.7.4 Comparison of TDMNN with DWT-SPIHT Technique

The results obtained for the TDMNN are compared with JPEG (DWT-SPIHT) compression technique. In this work software model for JPEG technique is also developed. Discrete Wavelet Transforms (DWT) and Set Partitioned Integer Hierarchical Tree (SPIHT) encoder techniques have been used to develop JPEG compression and decompression model. Images that have been considered for linear and hybrid networks have been considered for JPEG technique. Table 4.7 presents the results obtained and comparison of all the three techniques discussed.

Quality metrics for three different techniques have been compared and Fig. 4.19 is used for detailed analysis and observations. In Fig 4.19(a) bpp vs. MSE for baboon is presented for all the three techniques. Lower bpp implies the number of neurons in the hidden layer is less than the number neurons in the output layer. In this example, 16 pixels are compressed to 1 pixel which is 0.5 bpp, 16 pixels compressed to 2 pixels is 1 bpp. In the network compression data from 16 pixels to 1 pixel, is achieved by a single neuron in the hidden layer, hence all the 16 inputs are multiplied by the weight matrix of the single neuron network, thus exploiting the pixel to pixel correlation. As the number of neurons in the hidden layer increases, pixel to pixel dependency reduces and hence affects the MSE and PSNR. Thus it is observed that lower the bpp better is the network performance in case on neural network architecture.

Table 4.7 Comparison of quality metrics for linear, hybrid and DWT-SPIHT techniques

bpp	0.5	2	4	5	7.5
baboon	MSE				
DWT-SPIHT	715.9	183.8	46.80	14.62	1.97
NN- Hybrid	0.06	0.10	2.50	4.79	4.70
NN- Linear	1.09	1.41	2.94	4.47	6.33
baboon	PSNR				
DWT-SPIHT	19.58	25.48	31.42	36.47	45.17
NN- Hybrid	59.95	57.85	44.14	41.32	41.40
NN- Linear	47.73	46.62	43.44	41.61	40.11
image1	MSE				
DWT-SPIHT	11.31	1.93	0.91	1.09	1.22
NN- Hybrid	0.15	0.42	15.72	18.40	19.84
NN- Linear	2.41	5.64	7.28	6.52	8.87
image1	PSNR				
DWT-SPIHT	37.59	45.26	48.53	47.75	47.26
NN- Hybrid	56.2	51.82	36.16	35.4	35.15
NN- Linear	44.30	40.61	39.50	39.98	38.65

testim	MSE				
DWT-SPIHT	136.05	21.41	4.36	1.37	1.76
NN- Hybrid	6.16	68.73	34.22	26.30	60.38
NN- Linear	10.16	12.13	43.06	43.02	44.24
testim	PSNR				
DWT-SPIHT	26.79	34.80	41.73	46.75	45.65
NN- Hybrid	40.23	29.75	32.78	33.93	30.32
NN- Linear	38.0	37.29	31.78	31.79	31.67
peppers	MSE				
DWT-SPIHT	44.31	12.16	4.12	1.25	1.64
NN- Hybrid	0.10	27.09	47.06	47.25	53.02
NN- Linear	9.95	26.08	32.79	38.23	34.82
peppers	PSNR				
DWT-SPIHT	31.66	37.27	41.97	47.14	45.97
NN- Hybrid	57.83	33.80	31.40	31.38	30.80
NN- Linear	38.15	33.96	32.97	32.30	32.71
trees	MSE				
DWT-SPIHT	130.62	32.01	4.57	1.66	1.70
NN- Hybrid	0.01	39.48	1.20	57.23	68.40
NN- Linear	17.05	29.56	49.75	32.38	20.96
trees	PSNR				
DWT-SPIHT	26.97	33.07	41.52	45.90	45.82
NN- Hybrid	66.55	32.10	47.31	30.55	29.77
NN- Linear	35.81	33.42	31.16	33.02	34.91
pears	MSE				
DWT-SPIHT	33.70	9.40	1.48	0.62	1.71
NN- Hybrid	9.46E-04	57.05	58.19	67.17	67.18
NN- Linear	12.14	6.03	41.86	17.35	33.90
pears	PSNR				
DWT-SPIHT	32.84	38.37	46.41	50.17	45.78

NN- Hybrid	78.37	30.56	30.48	29.85	29.82
NN- Linear	37.28	40.32	31.91	35.73	32.82
Saturn	MSE				
DWT-SPIHT	2.43	0.40	0.70	0.71	0.71
NN- Hybrid	1.47	4.76	16.77	22.02	24.26
NN- Linear	2.61	10.31	6.16	10.03	8.63
Saturn	PSNR				
DWT-SPIHT	44.26	52.10	49.67	49.60	49.59
NN- Hybrid	46.44	41.34	35.88	34.70	34.28
NN- Linear	43.95	37.99	40.23	38.11	38.76
kids	MSE				
DWT-SPIHT	9.97	2.40	0.41	0.81	1.24
NN- Hybrid	3.02	9.96	49.80	22.95	45.20
NN- Linear	19.20	15.30	20.32	20.02	17.20
kids	PSNR				
DWT-SPIHT	38.14	44.31	51.99	49.03	47.16
NN- Hybrid	43.33	38.14	31.15	34.52	31.57
NN- Linear	35.29	36.27	35.04	35.11	35.75
rice	MSE				
DWT-SPIHT	199.70	44.37	13.70	4.31	1.57
NN- Hybrid	0.08	0.96	73.70	89.31	95.40
NN- Linear	8.59	17.80	22.03	21.80	24.60
rice	PSNR				
DWT-SPIHT	25.12	31.65	36.74	41.77	46.16
NN- Hybrid	58.76	48.27	9.45	28.60	28.33
NN- Linear	38.78	35.62	4.70	34.74	34.20

From Fig. 4.19(b), proposed technique outperforms JPEG technique at lower bpp, and also it is found that hybrid network achieves better PSNR compared to linear network. At higher bpp, JPEG based technique outperforms neural network techniques.

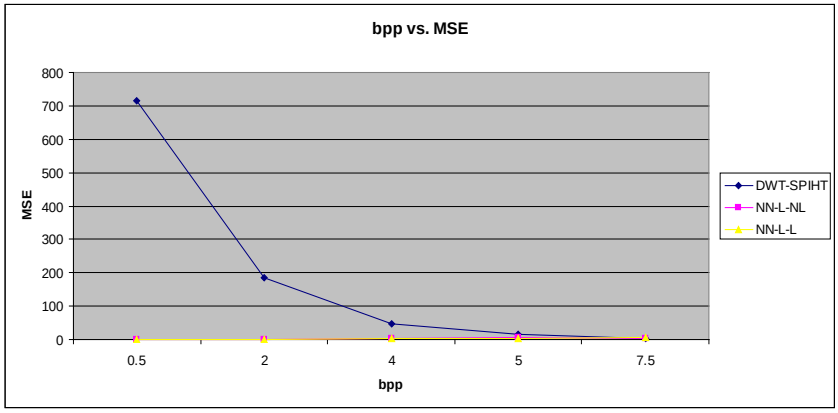


Figure 4.19 (a) bpp vs. MSE

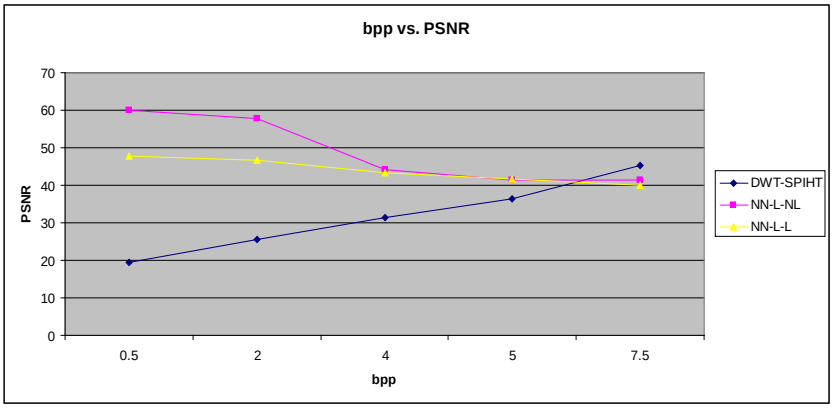


Figure 4.19 (b) bpp vs. PSNR

Figure 4.19 Comparison of quality metrics for all three techniques for baboon

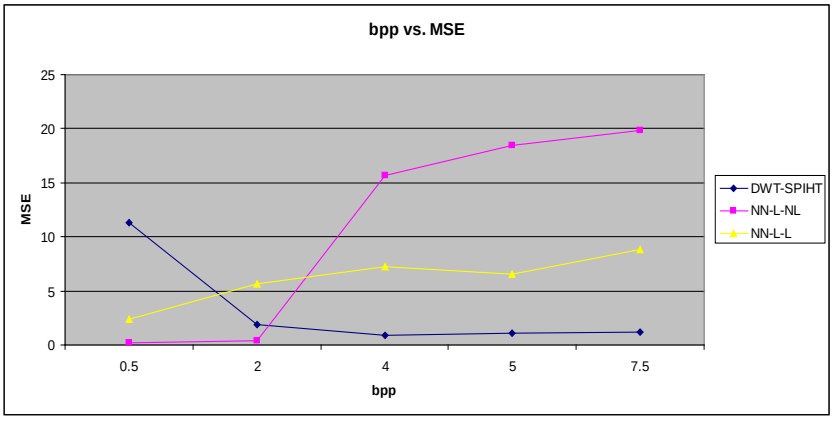


Figure 4.20 (a) bpp vs. MSE

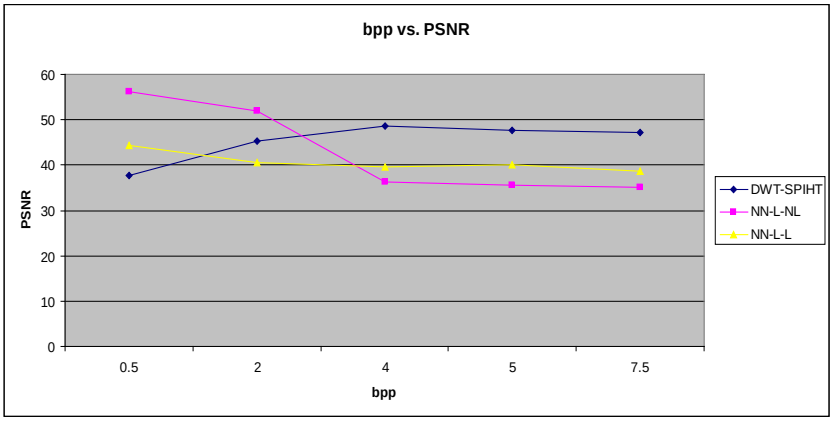


Figure 4.20 (b) bpp vs. PSNR

Figure 4.20 Comparison of quality metrics for all three networks for Image1

From Fig. 4.20(a), hybrid network achieves better MSE at low bpp, at higher bpp JPEG techniques achieves better results. For bpp 0.5 and 2 hybrid achieves better MSE. From Fig. 4.20(b), PSNR is higher for hybrid network for 0.5 bpp and 2 bpp compared with linear and JPEG based techniques. From Fig. 4.21, it is found that hybrid network performs better at lower bpp, at higher bpp JPEG is better than the other two techniques. Another very important observation made is that at higher bpp, linear networks perform better than hybrid networks. Also the PSNR is better for hybrid techniques only at lower bpp ($\leq$ 4 bpp). JPEG is better compared to the other two techniques at higher bpp ($>$ 4 bpp). Similarly, from the Table 4.7, we find that at lower bpp hybrid technique performs better than JPEG technique. As our focus is on use of neural network architecture, comparing hybrid and linear neural network technique, at higher bpp linear network architecture perform better than hybrid network. In table 4.7, MSE and PSNR for various bpp are highlighted. For each of the images considered in this work, best MSE and best PSNR for various bpp are highlighted using yellow color. From the results it is found that for 0.5 bpp hybrid networks achieves better PSNR and MSE compared to linear network. For 7.5 bpp, linear network performs better than hybrid network. For bpp between 2 and 5, for few of the images hybrid achieves better performance compared to linear network.

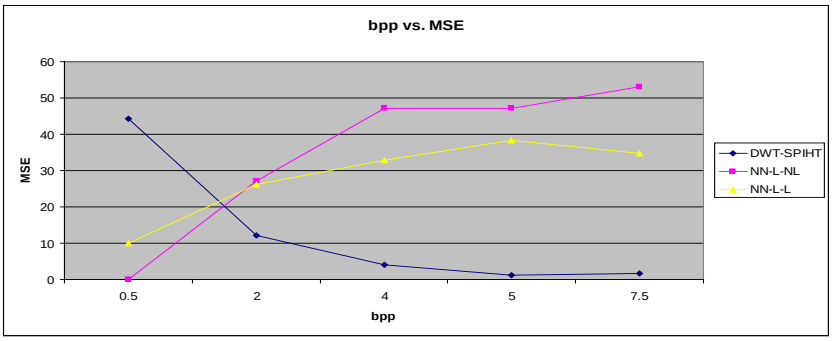


Figure 4.21 (a) bpp vs. MSE

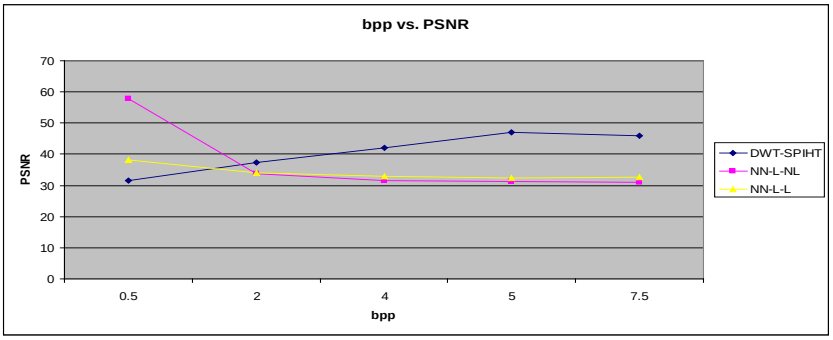


Figure 4.21 (b) bpp vs. PSNR

Figure 4.21 Comparison of quality metrics for all three networks for peppers

Computation time is another very important factor required to analyze network performance, which is discussed in the next section.

4.7.5 Comparison of Computation Time

MSE and PSNR have been used to compare the performances of the neural network architectures proposed in this work. Another very important factor that needs to be compared is the computation time of the network. The total time taken in compression and decompressing the image is considered as the computation time. In the proposed architecture, the input image is sub-divided, reordered and compressed using the hidden layer, and the compressed image is further decompressed using the output layer. The decompressed data is reordered and regrouped to the original image size. The total time taken to perform this process is considered as the computation time. The computation time for the proposed network is also compared with the total computation time using

JPEG technique. In JPEG technique, the input image is transformed using DWT and the transformed data is encoded using SPIHT technique. The encoded data is decoded and inverse transformed to decompress the compressed image. Table 4.8 presents the computation time for three different images with varying bpp. The results presented in the table are plotted in graphical form and is presented in Fig. 4.22. Computation time is measure in terms of the CPU time required to perform the operation. Computation time of DWT-SPIHT is higher compared to the time required by the network architectures proposed in this work. For bpp from 0.5 to 7.5 the computation time using the proposed technique is almost remaining constant.

Table 4.8 Computation time with variation in bpp

Time in seconds	**bpp**	**0.5**	**2**	**4**	**5**	**7.5**
Baboon	DWT-SPIHT	48.520	83.660	167.390	203.950	228.600
	NN - Linear	0.026	0.031	0.031	0.035	0.042
	NN - Hybrid	0.029	0.030	0.042	0.044	0.044
	bpp	**0.5**	**2**	**4**	**5**	**7.5**
Testim	DWT-SPIHT	57.240	95.600	189.360	192.560	191.880
	NN - Linear	0.027	0.027	0.028	0.030	0.030
	NN - Hybrid	0.027	0.030	0.30	0.037	0.037
	bpp	**0.5**	**2**	**4**	**5**	**7.5**
Image1	DWT-SPIHT	64.750	104.170	121.430	121.480	122.160
	NN - Linear	0.025	0.027	0.028	0.030	0.030
	NN - Hybrid	0.020	0.029	0.034	0.040	0.040

Computation time for JPEG is higher for increase in bpp. Fig. 4.22 depicts the variation of computation time with bpp for three different images. Computation time is

the total time required to compress and decompress the input image. Network training time is not considered, as the network is trained first and with the optimum weight and bias values TDMNN is constructed and is used for compression and decompression. Hybrid network is slower than linear network by almost 10 ms. In a linear network, network function is linear and hence there is no network function required. In a hybrid network, network function is Tansig, hence this introduces delay.

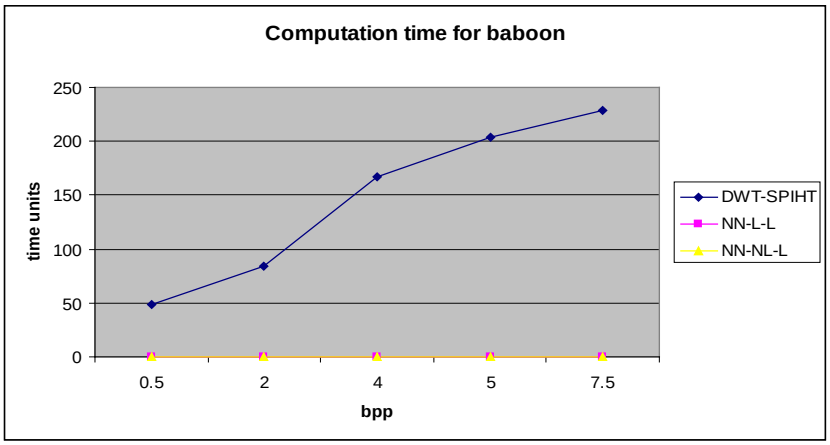


Figure 4.22 (a)

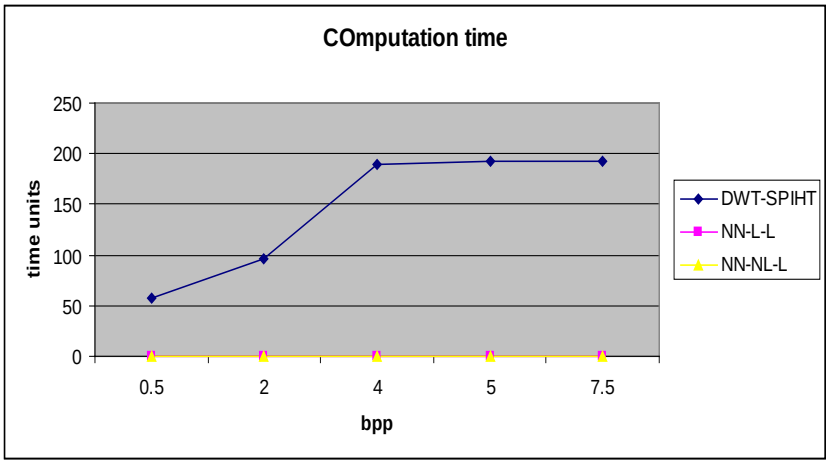


Figure 4.22 (b)

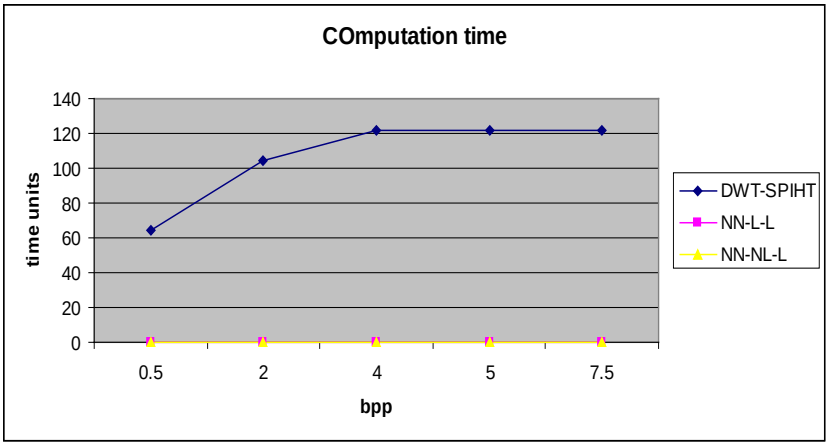


Figure 4.22 (c)

Figure 4.22 Computation time vs. bpp (a) baboon (b) testim (c) image1

Next section discusses the network performance for various images that were not part of training data sets. This analysis helps in generalizing the network architecture for image compression and decompression.

4.7.6 Network Performances for General Images

In this section images that were not part of the training sets have been used to compare the network performance. Fig. 4.23 shows the results of compression and decompression of test images. These images are not part of the training samples. These pictures have been taken as a test case to verify the network performance.

In order to understand the network architecture and its performance better, the following experiments have been carried out:

1. Impact of sub-block size on the network performance
2. Impact of multiple layers on network performance

In the next section the above two factors are discussed in detail based on the results obtained using software models developed.

This image has been removed

Figure 4.23 Results of 2-D multilayered neural network architecture

4.8 Analysis of Variations in Network Size

As discussed in previous sections, neural network architecture consists of multiple layers and each layer has multiple neurons. Number of layers and number of neurons in each layer influences the network performance for image compression. The number of weights and biases required to process data depends upon the network architecture. Analysis of optimum network size is presented in this section. Another analysis discussed in this section is the influence of image block size on the network performance. As discussed in

previous sections, images are sub-divided in smaller blocks and are compressed. The optimum block size that can give better quality metrics is analyzed.

4.8.1 Block Size Influences on Performance Metrics

Original image of size 256 x 256 was sub-divided into 4 x 4 sub-block and was compressed to 2 x 2 block using the two-dimensional multilayer network discussed in the previous section. In this section, network with various other input sub-block sizes are considered for analysis. Block sizes varying from 8 x 8 to 3 x 3 are considered for analysis. Table 4.9 presents the performance metrics variations with respect to input block size. The hidden layer size is fixed to 2 x 2, input layer and output layer sizes are varied as shown in Table 4.9. Compression Ratio (CR) is expressed in percentage.

Table 4.9 Input block size vs. Performance parameters for tree image

Block size	Maximum CR%	MaxError			MSE			PSNR		
		Tree	Pears	Peppers	Trees	Pears	Peppers	Tree	Pears	Peppers
8 x 8	98.43	127	165	97	394.50	267.90	100.00	22.17	23.80	28.13
7 x 7	97.95	136	134	100	258.38	155.70	81.26	24.00	26.20	29.03
5 x 5	96.00	123	149	86	203.50	112.97	58.43	25.04	27.60	30.46
4 x 4	93.75	109	131	78	147.80	79.00	38.74	26.43	29.14	32.24
3 x 3	88.88	93	106	75	111.35	60.68	32.47	27.66	30.30	33.01

From the results obtained and presented in Table 4.9 it is found that the MSE is best for block size of 4 x 4 and less. Hence in this work the block size selected is 4 x 4 block. The 2-D network designed also takes 4 x 4 matrixes and compresses the same to 2 x 2 sizes. Block size less than 4 x 4 has better MSE and PSNR, but affects the maximum compression ratio.

Fig. 4.24 to Fig. 4.26 compares the variations in block size with quality metrics for three different images. Sub-block size of 4 x 4 is recommended as it achieves better PSNR and MSE compared to block sizes greater than 4 x 4. Smaller the sub-block size better is the network performance, but the number of sub-blocks for a given image increases with smaller sub-block size. From the results obtained and presented in this

section, sub-block of size 4 x 4 is chosen as a trade-off between network performance and computation time.

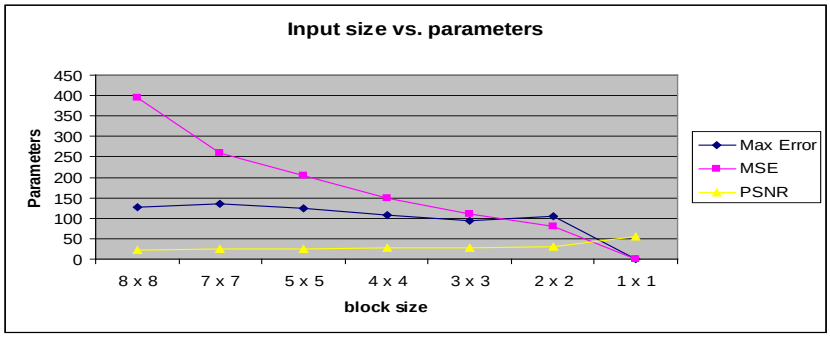


Figure 4.24 Input block size vs. performance parameters for trees

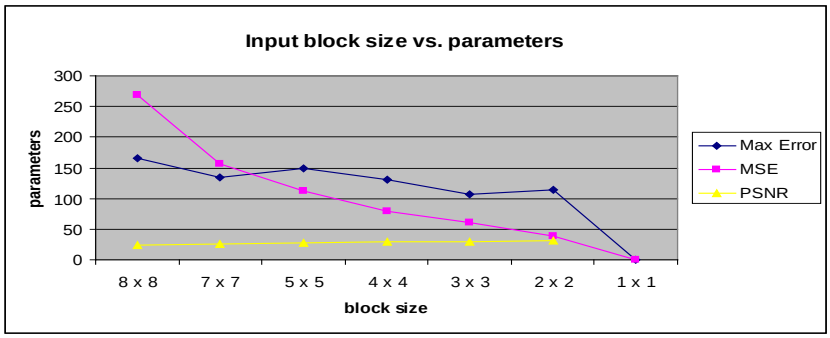


Figure 4.25 Input block size vs. performance parameters for pears

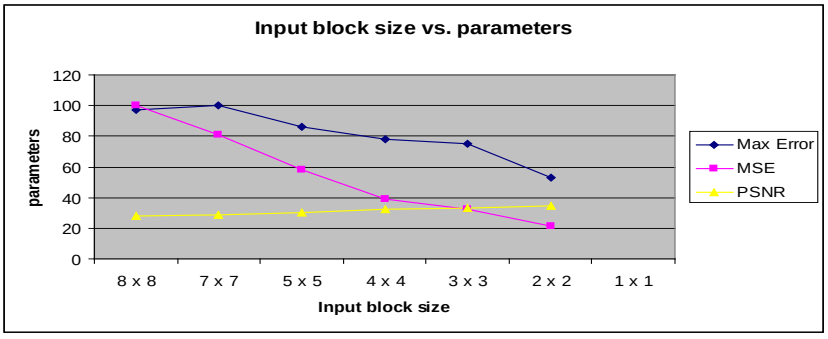


Figure 4.26 Input block size vs. performance parameters for peppers

Next section discusses the selection process for number of hidden layer to achieve better compression.

4.8.2 Impact of Multiple Layers on Network Performance

Another very important analysis that was necessary to carry out was the influence of number of layers on the performance parameters. In this section, the impact on performance parameters is analyzed by increasing the number of hidden layers. In this analysis input block size is fixed to 4 x 4, hence the numbers of inputs at the hidden layer is 16. Instead of compressing the input using one hidden layer, multiple hidden layers can also be used to compress the data. In this analysis, 16 inputs are compressed to 8 and then to 4, instead of 16 to 4 using one hidden layer. Table 4.10 and Table 4.11 show the performance parameters for three different images with multiple hidden layers. Values in brackets are for two layer network (hidden layer and output layer) network.

Comparing the results obtained for multiple hidden layers with single hidden layer presented in Table 4.6, the results clearly show that the network consisting of one hidden layer has better performance metrics. Hence in this research work, 2-D network with single hidden layer is selected for implementation.

Table 4.10 Performance parameters for 16:8:4:8:16 network

Parameters	Trees	Pears	Peppers
MSE	93.94 (49)	47.83(1.12)	25.59(0.06)
PSNR	28.00 (47)	31.00(49.00)	34.00(52.00)

Table 4.11 Performance parameters for 16:8:4:2:8:16 network

	Trees	Pears	Peppers
MSE	185(49)	115(01.12)	47(0.06)
PSNR	25(47)	27(49.00)	31(52.00)

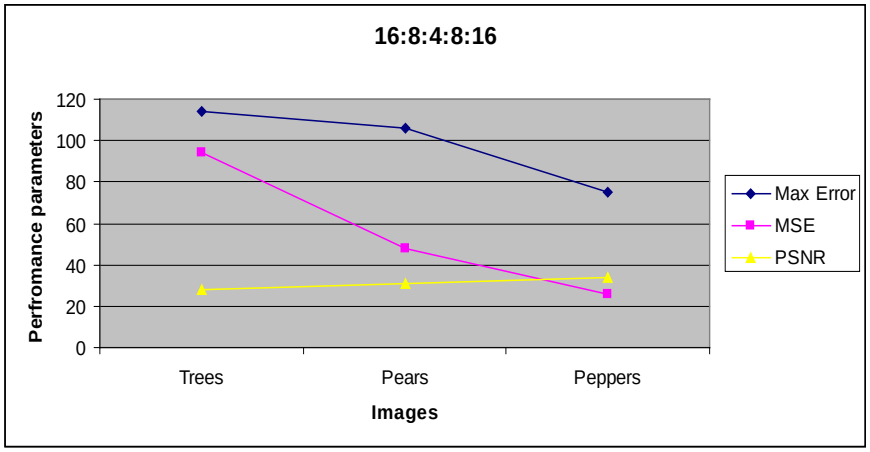


Figure 4.27 Performance metrics for multiple hidden layers

Fig. 4.27 and Fig. 4.28 presents the comparison of performance metrics for three images. From the simulations study carried out, another interesting observation is that the performance parameters of the network are not same for different images. There is a variation in MSE, PSNR and Maximum error with variation in input image properties. The performance of the network architecture is a function of image properties, unlike conventional techniques. In order to understand the network performance better (discussed in section 4.10), image properties are analyzed based on which the network architecture is modified to improve performances.

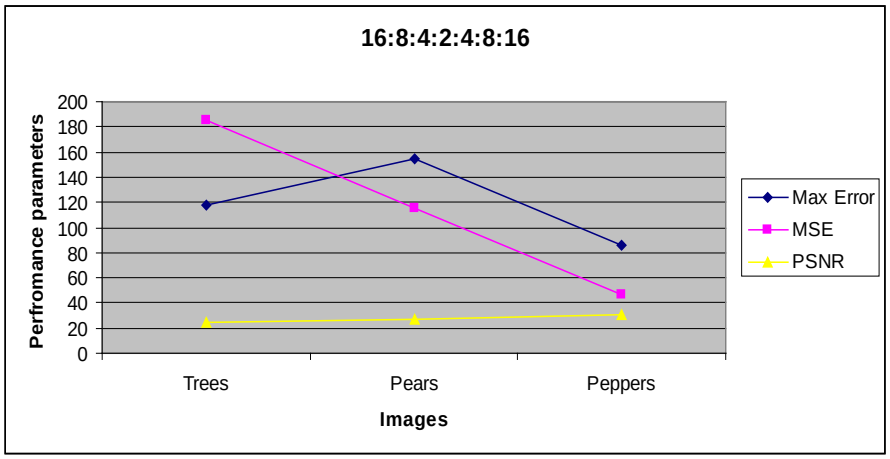


Figure 4.28 Performance metrics for multiple hidden layers

4.9 Noise Analysis and Error Analysis

One of the major objectives of this research work is to analyze the network performance under the influence of noise and error. Noisy image is given as input to the network, and the network performance is analyzed. Different noise sources such as Gaussian noise, Poisson noise and Salt & pepper noise with SNR of 10 dB are added to the image prior to compression and decompression. Error analysis is also carried out by introducing error in the compressed data.

4.9.1 Noise Analysis

TDMNN is trained to reproduce images even in the presence of noise. Test images are added with noise and set as input. Network needs to reproduce images without noise. Hence the target is set to images without noise. The network is trained for 100 epochs. The network learns to reproduce the original image from the noisy input data during the training phase. Optimum weight and biases are identified during the training phase. The network performance metrics such as MSE and PSNR are obtained for 0.5 and 1 bpp for four different images. The results obtained are compared with DWT-SPIHT technique. Images corrupted with noise (Gaussian, Poisson, and Salt & Pepper) are compressed and decompressed using TDMNN architecture and DWT-SPIHT technique. The results are tabulated in Table 4.12 to Table 4.14.

Table 4.12 presents the MSE and PSNR results obtained for four images with 0.5 bpp and 1 bpp compression. The results are obtained on images of size 256 x 256. From the results obtained the following are the observations made:

1. Without noise hybrid neural network architecture and linear neural network architecture achieve better MSE and PSNR. With salt and pepper noise added to the image, MSE and PSNR achieved using DWT-SPIHT technique have large variations as compared with hybrid and linear neural network architectures.
2. Neural network technique (hybrid and linear) achieve good MSE and PSNR even when the image is corrupted with noise.

Table 4.12 Results of noise analysis (Salt and Pepper)

Bpp = 1, Salt & Pepper Noise						
Mean Square Error						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With Noise	Without noise	With noise	Without noise	With noise
Baboon	619.07	687.93	482.46	627.34	868.21	1.40E+03
Testim	189.00	262.07	112.74	275.90	213.34	1.01E+03
Peppers	142.01	212.22	96.68	244.47	189.37	946.31
Image1	33.87	96.20	18.24	160.17	46.78	882.80
PSNR						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With noise	Without noise	With noise	Without noise	With noise
Baboon	20.21	19.75	21.29	20.15	18.78	16.68
Testim	25.36	23.94	27.60	23.72	21.28	18.09
Peppers	26.60	24.86	28.27	24.24	24.53	18.37
Image1	32.83	28.29	35.51	26.08	22.61	18.67
Bpp = 0.5, Salt & Pepper Noise						
Mean Square Error						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With Noise	Without noise	With noise	Without noise	With noise
Baboon	640.76	859.96	611.52	685.15	721.08	1.73E+03
Testim	177.60	438.96	171.82	251.67	194.90	1.44E+03
Peppers	139.74	383.19	132.75	206.78	145.50	1.20E+03
Image1	33.12	241.57	30.11	95.76	46.15	1.01E+03
PSNR						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With noise	Without noise	With noise	Without noise	With noise
Baboon	20.06	18.78	20.26	19.77	19.55	15.75
Testim	25.63	21.70	25.78	24.12	22.82	16.55
Peppers	26.67	22.29	26.90	24.97	21.50	17.33
Image1	32.92	24.30	33.34	28.31	30.90	18.07

Table 4.13 and Table 4.14 present the results obtained with Poisson and Gaussian noise.

Table 4.13 Results of noise analysis (Poisson)

Bpp = 1, Poisson Noise						
Mean Square Error						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With Noise	Without noise	With noise	Without noise	With noise
Baboon	619.07	628.18	482.46	500.11	868.21	928.65
Testim	189.00	195.36	112.70	127.34	213.34	247.44
Peppers	142.00	150.29	96.60	113.90	189.37	200.51
Image1	33.87	44.90	18.24	39.10	46.78	183.01
PSNR						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With noise	Without noise	With noise	Without noise	With noise
Baboon	20.21	20.15	21.29	21.14	18.78	18.94
Testim	25.36	25.22	27.60	27.08	21.28	20.75
Peppers	26.60	26.36	28.27	27.56	24.53	22.10
Image1	32.83	31.60	35.51	32.20	22.61	20.50
Bpp = 0.5, Poisson Noise						
Mean Square Error						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With Noise	Without noise	With noise	Without noise	With noise
Baboon	640.76	668.61	608.70	616.04	721.08	832.01
Testim	177.69	200.69	171.14	176.77	194.90	221.62
Peppers	139.74	169.53	132.20	139.19	145.50	200.46
Image1	33.12	66.47	30.04	38.70	46.15	185.71
PSNR						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With noise	Without noise	With noise	Without noise	With noise
Baboon	20.06	19.87	20.28	20.23	19.55	18.92
Testim	25.63	25.10	25.79	25.65	22.82	24.67
Peppers	26.67	25.83	26.91	26.69	21.55	25.11
Image1	32.92	29.90	33.30	32.24	30.95	25.44

Table 4.14 Results of noise analysis (Gaussian)

Bpp = 1, Gaussian						
Mean Square Error						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With Noise	Without noise	With noise	Without noise	With noise
Baboon	619.07	666.81	482.46	582.99	868.21	1.30E+03
Testim	189.00	233.42	112.74	212.65	213.34	861.29
Peppers	142.01	188.17	96.68	199.33	189.37	804.83
Image1	33.87	86.23	18.24	119.16	46.78	743.92
PSNR						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without Noise	With noise	Without noise	With noise	Without noise	With noise
Baboon	20.21	19.89	21.29	20.47	18.78	16.97
Testim	25.36	24.44	27.60	24.80	21.28	18.77
Peppers	26.60	25.38	28.27	25.13	24.53	19.07
Image1	32.83	28.77	35.51	27.36	22.60	19.41
Bpp = 0.5, Gaussian						
Mean Square Error						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With Noise	Without noise	With noise	Without noise	With noise
Baboon	640.76	808.32	608.70	649.61	721.08	1.55E+03
Testim	177.6	328.80	171.14	211.97	194.90	870.02
Peppers	139.7	300.99	132.20	175.05	145.50	812.92
Image1	33.12	196.99	30.04	71.69	46.15	744.18
PSNR						
	Hybrid		**Linear**		**DWT-SPIHT**	
	Without noise	With noise	Without noise	With noise	Without noise	With noise
Baboon	20.06	19.05	20.28	20.00	19.55	16.22
Testim	25.63	22.96	25.79	24.80	22.82	18.73
Peppers	26.67	23.34	26.91	25.69	21.55	19.03
Image1	32.92	25.18	33.35	29.576	30.95	19.40

Neural network is trained with the images without noise, and the training is carried out until the error is 0.001 of 100 epochs are carried out. In order to further increase the efficiency of the network, the training parameters can be changed. PSNR and

MSE are dependent on image. Hence, the network needs to be trained efficiently to meet the requirements of all the images.

4.9.2 Error Analysis

In this work, error analysis is also carried out to analyze the network performance. Error refers to bit errors occurring on the compressed stream of data. In DWT-SPIHT technique, the input image having 8 bpp is compressed to less than 8 bpp achieving compression. For analysis purpose, 0.5 bpp and 1 bpp is selected. The compressed stream of data using DWT-SPIHT technique is achieved in two steps. First image is transformed using DWT into sub-bands and in the second stage the sub-bands are encoded into bit stream using SPIHT. The compressed data is assumed to be transmitted, and due to transmission, bit error occurs. In this work, error is introduced randomly on the compressed data at various positions. For an image of size 256 x 256 selected for analysis the compressed bit stream size at 0.5 bpp is 32,768. The bit stream consisting of '1s' and '0s' are corrupted by changing the values at various bit positions. For example, at bit position 60, if the value is '1', it is changed to '0' or if '0' is changed to '1'. Fig. 4.29 presents the visual pictures of reconstructed output with and without bit errors. Original image of size 256 x 256 consisting of 524,288 bits are compressed to 32, 768 bits and is further decompressed to 524,288. The decompressed image is shown in Fig. 4.29(b).

Figure 4.29 (a) Input Figure 4.29 (b) Without Error Figure 4.29 (c) With error

Figure 4.29 Error analysis for Baboon image with 0.5 bpp, and error at 60th bit position

MSE and PSNR for this image is 721.08 and 19.55. Bit error is introduced randomly at the 60th bit position, at 60th bit position the bit value was '1', this was changed to '0', based on this error the image was reconstructed and is shown in Fig. 4.29 (c). The error at the 60th bit position introduces complete distortion in the decompressed data. MSE and PSNR for the decompressed image with error in 60th bit position is 14860 and 6.4107 respectively. Similar observations were made by introducing errors at various bit positions, every image was found to be distorted. Bit errors had larger influence on the reconstructed image. The results for other images are also presented in Table 4.15, MSE and PSNR have been obtained by introducing errors at 60th bit position only. Fig. 4.30 to Fig. 4.32 presents the results obtained using DWT-SPIHT technique for four different images. Images have been compressed at 0.5 bpp and error is introduced at the 60th bit position.

Table 4.15 Error analysis for DWT-SPIHT technique

DWT-SPIHT Technique, 0.5 bpp, Error at 60th bit position								
	Baboon		Testim		Peppers		Image1	
	Without Error	With error	Without Error	With error	Without Error	With error	Without Error	With error
MSE	721.08	14860	134.93	8860	45.5	10500	26.15	2050
PSNR	19.55	6.4100	26.82	8.6531	31.5	7.89	33.96	15.01

Figure 4.30 (a) Input Figure 4.30 (b) Without Error Figure 4.30 (c) With error

Figure 4.30 Error analysis for Testim image with 0.5 bpp, and error at 60th bit position

Figure 4.31 (a) Input Figure 4.31 (b) Without Error Figure 4.31 (c) With error

Figure 4.31 Error analysis for Testim image with 0.5 bpp, and error at 60th bit position

These images have been removed

Figure 4.32 (a) Input Figure 4.32 (b) Without Error Figure 4.32 (c) With error

Figure 4.32 Error analysis for Test image with 0.5 bpp, and error at 60th bit position

From the results obtained and shown in Table 4.15 and Fig. 4.30 to Fig. 4.32, the following are the observations made:

1. Without error in the compressed data, MSE and PSNR for all the images have different values, this is due to the fact that all the images have different information and hence the compression scheme exploits this information and compress data.

2. PSNR for Image1 is higher compared to PSNRs for other images at 0.5 bpp. This is due to the fact that Image1 does not have large variations in the background. At 0.5 bpp, reconstructed images have lost sharpness and are blurred.
3. Error at the 60^{th} bit position corrupts the reconstructed image completely. Only in case of Image1, the distortion is not very severe and visual information is retained. This is due to the fact that, the DWT output transposes the temporal information into multiple sub-bands and each sub-band holds the low frequency and high frequency components of the original image at different band levels. The significant components in the different sub-bands are encoded using SPIHT encoding scheme. Bit error occurring at any bit positions influences the decoding of the significant values at various sub-band levels and hence the image gets distorted. As there is interdependency between various sub-band levels which is captured by the SPIHT technique, any bit error affects the reconstructed image.
4. Reconstructed images with and without error have very large variations in MSE and PSNR as shown in Table 4.15. For example, for baboon image PSNR of 19.55 changes to 6.4107, similar changes are observed in all other images.
5. From the results obtained, it is concluded that bit errors impact reconstruction process in DWT-SPIHT technique. Similar, observations were for other images, by inserting errors at various other positions.

In case of neural network based approach, input pixels are multiplied by the weight elements and added with bias to compress the data. In this work, image of size 256 x 256 is sub-divided into 4 x 4 block. Each 4 x 4 block is rearranged to 16 x 1 matrix sizes, for the entire 256 x 256 image the rearrangement matrix size is 16 x 4096. The network after training consists of 2 x 16 weight matrix and 2 bias values to achieve 1 bpp. To achieve 0.5 bpp, the weight matrix is 16 x 1 and 1 bias value. The decompressor unit in case of 1 bpp consists of 2 x 16 weights and 16 bias values. For 0.5 bpp, the weight matrix is 16 x 1 and has 16 bias values. Four different images are considered for error analysis. The reordered image is compressed using the compressor unit. Compressed data is of size 2 x

4096 and 1 x 4096 for 1 bpp and 0.5 bpp respectively. In the neural network approach, compression is achieved in one step, there is no encoding stage. Error is introduced by changing the compressed data values. Fig. 4.33 shows the snap shot of a compressed image data. Only a partial data is presented for the sake of understanding the procedure of error analysis. In order to introduce error on this compressed data, the decimal values at different positions are modified. For example, in Fig. 4.33 the compressed data is stored in a variable name **atest41**, this consists of 4096 pixels. The decimal value at 3rd position having a value -1.2351 is modified and changed to 0. In this work, the compressed data is modified at positions 200 to 210. 11 decimal positions from 200 to 211 are changed to 0. The modified compressed data is presented to the decompressor unit for reconstruction.

```
>> size(atest41)

ans =

           1        4096

>> atest41

atest41 =

  Columns 1 through 12

   -1.3340   -1.3676   -1.2351   -1.1291   -1.0709   -1.2064   -1.0766   -1.0788   -1.0483   -1.1354

  Columns 13 through 24

   -1.1986   -1.0933   -1.1181   -1.1945   -2.6855   -3.7644   -3.5407   -3.2599   -3.2359   -3.5556

  Columns 25 through 36

   -4.3661   -4.1517   -3.4635   -2.6230   -2.2618   -1.8731   -1.4085   -1.4177   -1.5032   -1.7189
```

Figure 4.33 Compressed output

Fig. 4.33 presents the reconstructed output for all four images compressed at 0.5 bpp using linear network. Fig 4.34 presents the reconstructed output for all four images with 1 bpp using hybrid network. For all the images the error is introduced between 200th decimal positions to 210th decimal position. From the obtained results if is observed that the error introduced at the given positions corrupt the data only at appropriate positions as highlighted in yellow circles. The errors do not corrupt the entire image as in DWT-SPIHT technique. Image quality is not affected and most of the information in the original image is retained in the reconstructed image, even in the presence of error. Errors

positions have been changed and analysis is carried out. It is observed that errors introduced at any decimal positions do not corrupt the entire image but introduce disturbance only at specific locations. Multiple images were used for error analysis and results obtained are not presented. It is observed that error introduced on the compressed data does not impact the information content in the reconstructed data. Fig. 4.34 to Fig. 4.37 presents the reconstructed images using linear and hybrid network. Table 4.16 and Table 4.17 presents the image quality metrics for linear and hybrid network.

Figure 4.34 Reconstructed output at 1bpp using linear network

Figure 4.35 Reconstructed output at 0.5 bpp using linear network

Figure 4.36 Reconstructed output at 1 bpp using hybrid network

These images have been removed

Figure 4.37 Reconstructed output at 0.5 bpp using hybrid network

Table 4.16 Error analysis for linear network

MSE, 1 bpp Linear Network				
	Baboon	Testim	Peppers	Image1
Without error	473.59	107.05	102.35	18.99
With error	502.19	125.95	121.46	39.54
PSNR, 1 bpp Linear network				
	Baboon	Testim	Peppers	Image1
Without error	21.37	27.83	28.02	35.34
With error	21.12	27.12	27.28	32.16
MSE, 0.5 bpp Linear Network				
	Baboon	Testim	Peppers	Image1
Without error	610.9	171.12	132.47	30.07
With error	616	176.89	140.01	33.73
PSNR, 1 bpp Linear network				
	Baboon	Testim	Peppers	Image1
Without error	20.27	25.78	26.9	33.34
With error	20.17	25.69	26.81	32.85

Table 4.17 Error analysis for hybrid network

MSE, 1 bpp Hybrid network				
	Baboon	Testim	Peppers	Image1
Without error	624.39	176.53	136.34	31.36
With error	648.49	195.07	154.36	50.24
1 bpp Hybrid network				
	Baboon	Testim	Peppers	Image1
Without error	20.17	25.66	26.78	33.16
With error	20.01	25.22	26.24	31.11
MSE, 0.5 bpp Hybrid network				
	Baboon	Testim	Peppers	Image1
Without error	610.30	172.80	133.51	30.69
With error	611.52	174.41	133.67	31.07
PSNR, 0.5 bpp Hybrid network				
	Baboon	Testim	Peppers	Image1
Without error	20.27	25.75	26.87	33.25
With error	20.26	25.71	26.87	33.20

In order to minimize the impact of channel error on the reconstructed image, error control codes are used to encode the compressed data.

After compression of image using DWT-SPIHT, the compressed data can be encoded using error control codes. Error control codes are used to detect errors and correct errors when the compressed data is transmitted in a communication channel that has noise. In order to detect and correct errors, the encoder transmits redundant information along with compressed data. At the receiver, the decoder uses the redundant information to detect and correct the errors that has occurred during transmission. There are two types of error-control coding techniques, block codes and convolution codes. Block coding techniques map a fixed number of message symbols to a fixed number of code symbols. A block coder treats each block of data independently and is a memoryless device. Bose-Chaudhuri-Hocquenghem (BCH), Low-density parity-check (LDPC), cyclic, hamming and Reed Solomon (RS) are different types of block codes (Wicker, S. B 1995). In this work, RS encoder is used as error control codes to detect and correct errors. Reed-Solomon codes use m-bit symbols instead of bits. A message for an [n, k] Reed-Solomon code must be a k-column Galois array in the field GF(2^m). Each array

entry must be an integer between 0 and 2^m-1. The code corresponding to that message is an n-column Galois array is GF(2^m). The codeword length n must be between 3 and 2^m-1. The quantities n and k are input parameters for Reed-Solomon code. In RS encoder, m is number of bits per symbol (3 to 16), n is number of symbols per codeword, k is number symbols per message and t is error correction capability ((n-k)/2).

In this work, [255,239] RS encoder is used to encode the compressed data and error is introduced on the encoded data. At the receiver, the encoded data along with noise is decoded using RS decoder. The compressed bits were converted to symbols of each 8-bit, and symbols were grouped into frames of 225 symbols. Each block of symbols were encoded using RS encoder, error were introduced on the encoded data. From the simulation results obtained, it is found that due to introduction of error control codes, maximum of 4 symbol errors were corrected by [255,239] RS encoder. Errors occurring on more than 4 symbols, were not be corrected by the RS encoder. Thus, for burst error correction of compressed data RS encoder would improve the image quality for a maximum of (n-k)/2 errors. DWT-SPIHT based image compression algorithm along with RS encoder is capable of detecting and correcting error, however, the processing time is increased by 9 seconds due to RS encoder logic.

From the results obtained and presented in Table 4.16 and Table 4.17 the following are the observations made:

1. Linear network are better than hybrid network in reconstructing the images considered from the compressed data.
2. MSE and PSNR variations with and without error are very less, as compared with DWT-SPIHT technique.
3. At 0.5 bpp hybrid network exhibits better performance in terms of MSE and PSNR compared with 1 bpp.
4. Use of error control codes to minimize channel errors in DWT-SPIHT gives improvement in image reconstruction at the cost of processing delay. TDMNN

technique is capable of reconstructing image even in the presence of channel error and is faster compared with DWT-SPIHT.

5. Careful observation of the reconstructed data, it is found that checker blocks occur in the reconstructed image. This is due to the fact that in the software reference model, image is sub-divided into 4 x 4 blocks and hence this introduces checker block error. This is one of the major limitations of NN based compression technique as compared with DWT-SPIHT technique. The checker block error can be minimized by combining DWT with Neural Network.

A paper titled TDMNN architecture for image compression and decompression is prepared based on the results obtained and is submitted to *International Journal on Signal Processing.*

The disadvantage of DWT with TDMNN is that the DWT decomposes the images into sub-bands, image information are represented in terms of low frequency and high frequency components and are available in each sub-band of the transformed data. SPIHT encoder encodes the significant and insignificant pixels using hierarchical approach (Rabbani, M and Joshi, R 2002). Every encoded bit has information of the original image; hence any error on the compressed data significantly affects the reconstructed image. In TDMNN due to the training process, the network is trained to reproduce original image even in the presence of noise or error, this advantage of TDMNN achieves better performance for the network as compared with DWT-SPIHT technique. However, in TDMNN technique, as the input image is sub-divided into sub-blocks, the reconstructed images have checker blocks as observed in Fig. 4.34 to Fig. 4.37. This is one of the limitations of TDMNN. In order to over-come this limitation, image is first transformed into sub-bands using DWT and the transformed sub-bands are compressed using the TDMNN architecture. DWT transformation extracts frequency components present in a given images at various sub-bands. The network is trained to learn this information content. Weights and biases are obtained for the network based on the image features. Also, as the TDMNN compresses 4 x 4 sub-bands to 2 x 2 outputs, compression

is achieved. Errors in channel do not impact the reconstruction process as TDMNN is trained to work in such environments. Further, the checker block error in the reconstructed output is eliminated as the images are not sub-divided into smaller sub-blocks. This technique not only eliminates checker block error, but also reproduces images even in the presence of error. DWT with neural networks have been proposed and used for image compression and decompression (Szu, Telfer and Garcia 1996, Szu, Telfer and Kadambe 1992, Veronin *et al*. 1992, Zhang 1996, Robinson and Kecman 2003 and Barni, Bartolini and Piva 2001). DWT combined with TDMNN architecture is designed, modeled and analyzed for image compression and decompression. DWT-combined with TDMNN architecture for image compression is further extended to 3D-DWT and TDMNN architecture. This architecture is documented by the author and is sent for patent review process. Based on the results and analysis carried out on TDMNN architecture the following are the conclusions:

1. Hybrid, linear and nonlinear TDMNN architecture have been designed, modeled and simulated.
2. 1-D backpropagation training algorithm have been modified and used to train TDMNN architecture.
3. Performances of TDMNN (Hybrid and Linear) architecture are compared with DWT-SPIHT technique for various images and compression ratios (bpp).
4. TDMNN network performance is a function of image. Based on the image properties, hybrid or linear TDMNN architecture is used for achieving better results.
5. Computation time is another metric that is computed for TDMNN and compared with DWT-SPIHT technique. TDMNN architecture is faster compared to DWT-SPIHT.
6. Two-layered network with sub-block size of 4 x 4 is the selected TDMNN architecture.
7. TDMNN architecture is immune to noise and also immune to channel errors.

Based on the conclusion presented above, next section discusses adaptive TDMNN architecture for image compression and decompression.

4.10 Adaptive TDMNN Architecture for Image Compression and Decompression

Based on the results obtained and presented in previous sections, as the network performance is image dependent, it is evident that use of hybrid or linear or nonlinear network would not be a viable solution for image compression and decompression that can be used to compress any image. Compression ratio (bpp) and computation time plays another important factor to be considered for image compression and decompression. Hence in this section, an adaptive TDMNN architecture is proposed, designed and analyzed for image compression and decompression. TDMNN performance is a function of input image and bpp, next section discusses the very basics of images and its properties. This discussion helps in understanding the need for adaptive image processing technique. Adjacent pixels in an image are highly correlated, this redundancy between pixels in spatial domain need to analyzed and compared with multiple images.

4.10.1 Linear Correlation in Spatial Domain

In order to understand the pixel to pixel correlation property, multiple images are considered. For convenient illustration, the image is cut into blocks of two pixels, in order to find the linear correlation between adjacent pixels, pixels blocks of size two is ordered into Cartesian coordinates. The results of this pixel-pixel relation are presented in this section for discussion. Fig. 4.38 shows the linear correlation between two adjacent pixels for multiple images. It is very clear that the every image has linear correlation between adjacent pixels. This correlation is exploited for image compression. If a straight line is drawn linking maximum number of pixels, the points on straight lines represents the approximation matrix using which the image properties can be represented. The weight matrix obtained during training tends to move toward this optimum point. From

the results presented in Fig. 4.38(a) to 4.38(h), it is clear that for most of the image maximum number of pixels are closer to this straight line. The slope of the straight line may vary, or the number of points available may spread across the straight line depending upon the information in the image.

Figure 4.38 (c) Printer

These images have been removed

Figure 4.38 Linear correlations between adjacent image pixels

From the results presented in Fig. 4.38, there exists high correlation among adjacent pixels in a given image. Images consisting of multiple edges and large variation in intensities in the spatial domain, as seen in Fig 4.38(e) to Fig. 4.38(h), there is a wide spread of pixel values. This spread in pixels defines the amount of information in a given image. Hadi and Jamzad (2009) have classified the information content in an image based on three parameters:

1. Entropy
2. Activity and
3. Pattern trajectory in blocks

Rahman and Rahman (2003) have proposed new architectures for image compression based on variations in neural network architectures. Based on the work reported by Hadi and Jamzad (2009) and Rahman and Rahman (2003), in this research adaptive TDMNN architecture is proposed for image compression and decompression. From the results obtained and analysis carried out, network performance is a function of image input and bpp. Hence, adaptive 2-D multilayered network is proposed for implementation.

4.10.2 Adaptive TDMNN

Hadi and Jamzad (2009) in their paper have reported that activity and pattern trajectory for a given image given accurate results in classifying images, but they consume more time. In this work, entropy factor is considered for classifying images. Entropy is calculated for every image before compression. Based on the entropy computed for a given image, compression ratio or bpp and required computation time, a suitable TDMNN architecture is selected for compression and decompression of images. In this architecture, computation time is also used to select the TDMNN architecture. Bpp and computation time are user inputs that needs to be defined during the compression and decompression process. A detailed discussion on Entropy is presented in Appendix- C. Fig. 4.39 presents the adaptive TDMNN architecture.

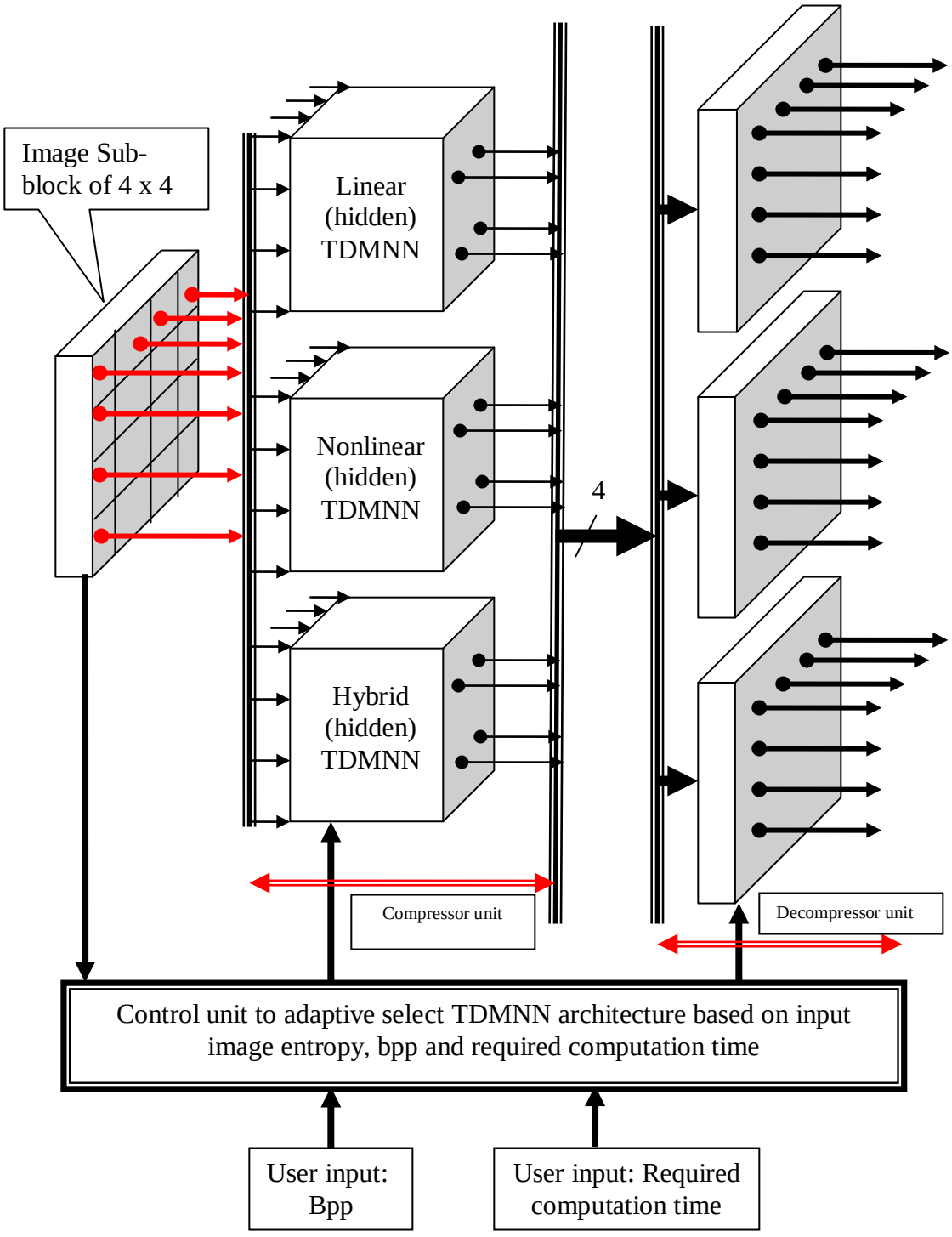


Figure 4.39 Adaptive TDMNN architecture

As shown in Fig. 4.39, input image of size 4 x 4 is given as input to any of the TDMNN architecture. The architecture consists of three different TDMNN architecture (hybrid, linear and nonlinear), based on user defined inputs (bpp and computation time) the control unit selects one of the TDMNN architecture for compression. For every sub-block of the input image entropy is computed, based on the entropy value and user defined inputs provided the control unit adaptively selects the TDMNN architecture. Entropy can be calculated for entire image or can be computed for every sub-block of the image. Entropy calculated for every sub-block of an image would be more accurate in choosing TDMNN architecture. The software reference model developed computes entropy for image which is used to adaptively select the required TDMNN. At the decompressor unit, the compressed data is decompressed by one of the TDMNN architecture correspondingly selected by the control unit. As the system is capable of choosing different TDMNN architectures to achieve better performance this architecture is called as Adaptive TDMNN architecture. Fig. 4.40 represents the flow chart for the proposed adaptive TDMNN architecture. Based on experimental results carried out the flow chart for the adaptive TDMNN architecture is designed and is presented in Fig. 4.40.

Entropy for a given image is computed, entropy values are classified into three classes.

- ⇒ Class 1: Entropy value - 6 to 7.5.
- ⇒ Class 2: Entropy value - 5 to 6.
- ⇒ Class 3: Entropy value - 4 to 5.

Higher the entropy, information content is high. From the discussion carried out from the previous section, for lower values of bpp hybrid TDMNN performance was found to be better than linear TDMNN. For higher values of bpp linear or nonlinear is better. Based on the results discussed in the previous section, flow chart for adaptive TDMNN is designed. Software reference model is developed and the network performance is analyzed for different test images.

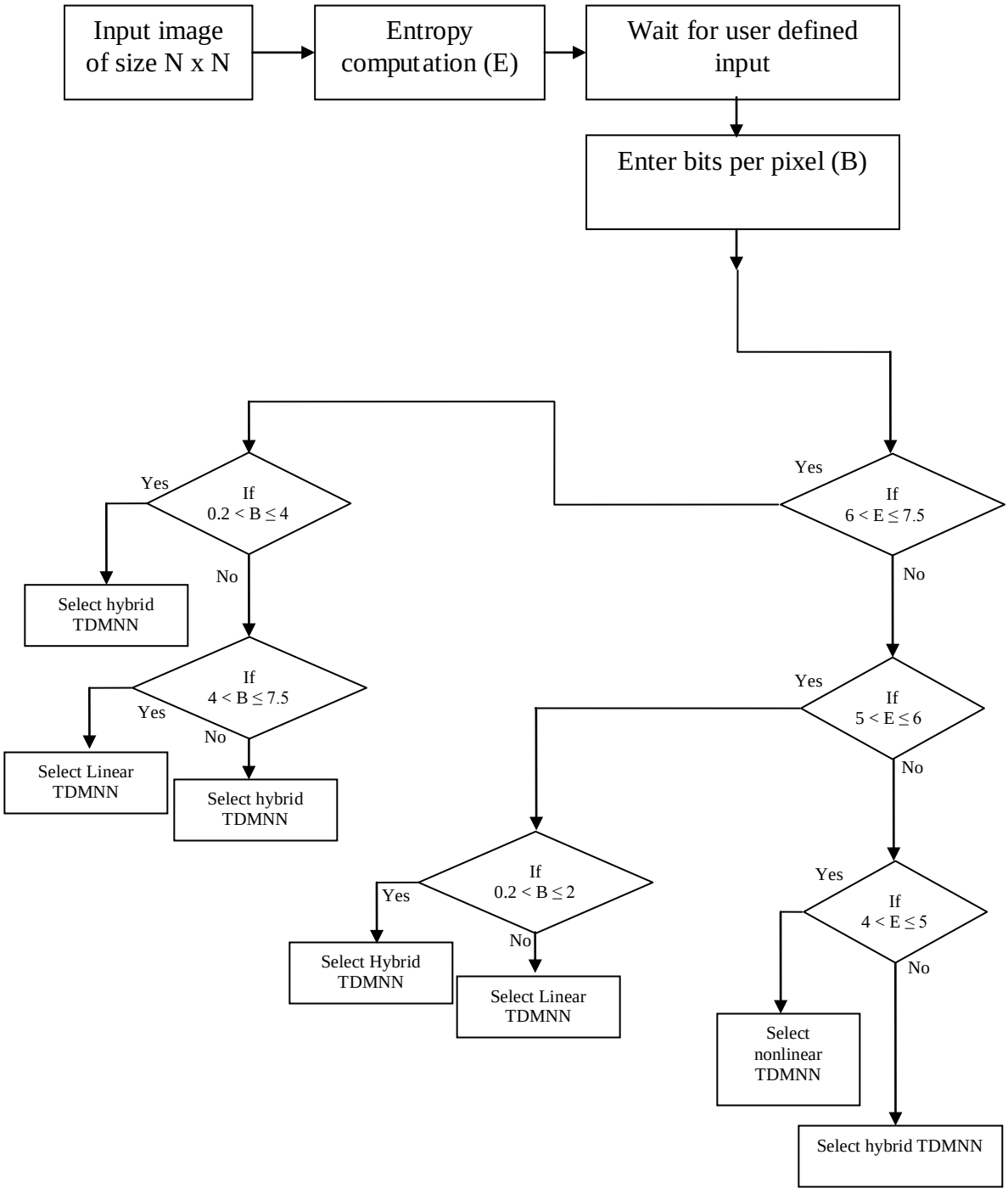


Figure 4.40 Flow chart for adaptive TDMNN architecture

Adaptive TDMNN architecture is developed based on the algorithm presented in Fig. 4.40. The default architecture for the proposed system is hybrid TDMNN. Depending upon user defined input, the network automatically selects corresponding architecture for compression and decompression of input image. Fig. 4.41 presents the entropy for 18 images. From the selected set of images, for which entropy is computed based on the equation in Appendix-C, entropy for given images vary from 4 to 8. Also based on the results presented in section 4.7, a control unit is designed to adaptively select the required TDMNN architectures based on image entropy and bpp.

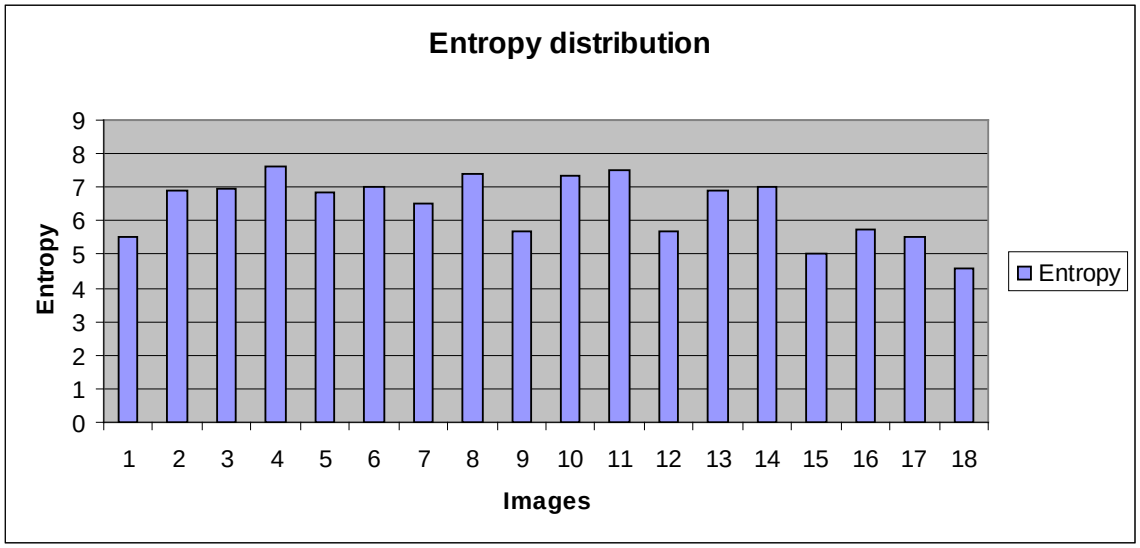


Figure 4.41 Entropy distributions of different images

Fig. 4.41 presents the entropy of multiple images considered for analysis in this work. The adaptive TDMNN architecture is analyzed for its performance for 9 different images. Entropy is calculated first based on the entropy and required bpp corresponding TDMNN architecture is selected for image compression as shown in Table 4.15. The compressed data is transmitted along with network code to the receiver, so that corresponding TDMNN is chosen for reconstruction. In this research work, hybrid, linear and nonlinear networks are designed to compress the image data. The network is trained using the software reference model. As shown in Table 4.18, based on results presented

in section 4.7, and entropy values are computed for each of the images, control unit is designed for adaptive TDMNN.

Table 4.18 Adaptive TDMNN selection guide

Images	Entropy	Selected Network		
		Bpp < 4	Bpp = 4	Bpp > 4
Baboon	7.4	Hybrid	Hybrid	Linear
Image1	6.4	Hybrid	Linear	Linear
Testim	7.1	Hybrid	Hybrid	Linear
Peppers	6.5	Hybrid	Linear	Linear
Trees	5.5	Hybrid	Linear	Linear
Pears	6.9	Hybrid	Hybrid	Linear
Rice	7.0	Hybrid	Hybrid	Linear
Saturn	5.6	Hybrid	Linear	Linear
Kids	4.8	Hybrid	Linear	Linear

Table 4.19 and Table 4.20, presents the results of MSE and PSNR for adaptive TDMNN architecture and is compared with TDMNN architecture. The compression ratio is fixed at 75%. The adaptive TDMNN computes entropy for the given image, and based on this information appropriate network is selected for compression and decompression. From the results presented in Table 4.19, MSE is better in case of adaptive TDMNN. Due to network adaptability, there is improvement in network performance compared to TDMNN architecture.

Table 4.19 Comparison of adaptive TDMNN with TDMNN

Images	Network Selected	PSNR (Adaptive TDMNN)	PSNR (TDMNN)
Trees	Nonlinear	30.9	27.4
Board	Hybrid	32.9	28.7
Circuit	Hybrid	37.8	33.0
Greens	Linear	46.3	36.0
Football	Hybrid	35.4	29.8
Cameraman	Hybrid	33.5	26.4
Tape	Hybrid	31.0	27.0
Fabric	Linear	33.9	26.8
Pears	Linear	32.0	29.0

Saturn	Nonlinear	28.3	24.0
Peppers	Linear	25.7	22.0
Tier	Hybrid	32.3	28.3
Kids	Nonlinear	23.5	18.4
Rice	Hybrid	27.5	24.6
Cell	Nonlinear	26.9	16.7

Table 4.20 Comparison of adaptive TDMNN with TDMNN

Images	Network Selected	MSE (Adaptive TDMNN)	MSE (TDMNN)
Trees	Nonlinear	21.0	22.9
Board	Hybrid	18.0	21.2
Circuit	Hybrid	16.9	18.0
Greens	Linear	12.7	14.7
Football	Hybrid	18.0	21.3
Cameraman	Hybrid	17.0	21.2
Tape	Hybrid	19.8	22.6
Fabric	Linear	15.0	16.2
Pears	Linear	14.3	17.8
Saturn	Nonlinear	27.0	32.0
Peppers	Linear	22.0	25.0
Tier	Hybrid	27.0	29.3
Kids	Nonlinear	38.0	42.0
Rice	Hybrid	25.4	26.0
cell	Nonlinear	45.7	71.0

Table 4.21 Adaptive TDMNN architecture comparison

	Barbara	
Quality metrics	DWT-SPIHT	Adaptive TDMNN
Max Error	139	168
MSE	736	447
PSNR	19	21
	Lena	
Quality metrics	DWT-SPIHT	Adaptive TDMNN
Max Error	125	123
MSE	562	202
PSNR	20	25

The results presented in Table 4.19 and Table 4.20 was for images that were part of training data set. In order to analyze the network performance for generic images, two images Lena and Barbara were considered. These images have been considered as they form the standard images for any image processing activity. The results obtained for these images are presented in Table 4.21, and are compared with DWT-SPIHT technique.

From the results obtained, it is concluded that proposed adaptive TDMNN is superior to DWT-SPIHT technique in compressing and decompressing images. For bpp 4 and less than 4 adaptive TDMNN architecture is recommended for image compression. Further, proposed architecture is immune to noise and also immune to channel errors. Superiority of the proposed network is due to the adaptability of the network as per the entropy of the input image and required bpp. Further, the network performance can be explored with variations in block size and make the network adaptable to input block size. Instead of computing entropy for the entire image, entropy for sub-blocks can be computed and the network can be made adaptable. This would further increase the accuracy of the network. However, the scope of this research is to realize the proposed adaptive TDMNN using VLSI technology. Chapter 5 presents a detailed discussion on VLSI implementation of proposed adaptive TDMNN architecture. The 2-D adaptive network proposed in this chapter achieves better results compared with the conventional technique. The adaptive network architecture consists of three types of 2-D network (Tansig-Purelin for hybrid networks, Tansig-Tansig for nonlinear networks and Purelin-Purelin for linear networks). The basic building blocks for the three architectures are multipliers, adders and network functions (Tansig function and Purelin function).

Chapter 5 – VLSI Implementation of Adaptive TDMNN Architecture

5.1 Introduction

This chapter discusses design, modeling and simulation of the adaptive two-dimensional multilayer neural network architecture proposed for image compression and decompression. This chapter also discusses implementation results and analysis of the proposed architecture for compression and decompression of images. The adaptive TDMNN architecture achieves better performance than the 2-D network and the conventional technique. The network is trained using multiple image data sets. The optimum weight matrix and bias elements obtained after training are used to compress images using adaptive approach. The network estimates the entropy for the given image and selects a suitable TDMNN for compression and decompression. Linear network, nonlinear network or hybrid network is selected based on entropy of image. The weight matrix and bias elements obtained during offline training process is converted into digital data and stored in memory. As per the analysis carried out in the previous chapters, image is sub-divided into 4 x 4 block size and is fed into the network. For every 4 x 4 blocks to be processed by the network, 4 x 16 weight matrix and 4 bias elements are required. For every 4 x 4 block to be compressed to 2 x 2 block, 64 multipliers (4 x 16), 64 adders (15 x 4 + 4) are required for the compressor block. Similarly, the decompressor requires 64 multipliers and 64 adders. Linear networks have Purelin functions and hence network functions are not required for hardware implementation. To realize nonlinear transfer function used in nonlinear network and hybrid network translinear circuits are required for hardware realization. Fig. 5.1 shows the top level architecture of the proposed network for hardware implementation.

5.2 Design Analysis

Video sequences are captured at a frame rate of 30 frames per second; every image consisting of 256 x 256 pixels would be captured and will be available on image sensors

for time duration of 33ms. The total amount of time available to process every frame is 33ms, within this time the proposed architecture should compute the entropy of the image, based on the entropy particular network need to be selected. Based on the network selected the image is sub-divided into 4 x 4 blocks. Assuming that the total time to preprocess the data requires 10 ms, for every 4 x 4 block to be processed the total amount of time available is 5.6 µs (256 x 256 has 4096 4 x 4 blocks, 23 ms is the processing time, this implies for 4 x 4 block total time is 23 ms / 4096). Within this time the network has to perform 64 multiplications and 64 additions at the compressor end and same number of computations at the decompressor end. Hence the circuit designed should meet the timing requirement, and the hardware circuits designed should be able to have a maximum delay of 5.6 µs.

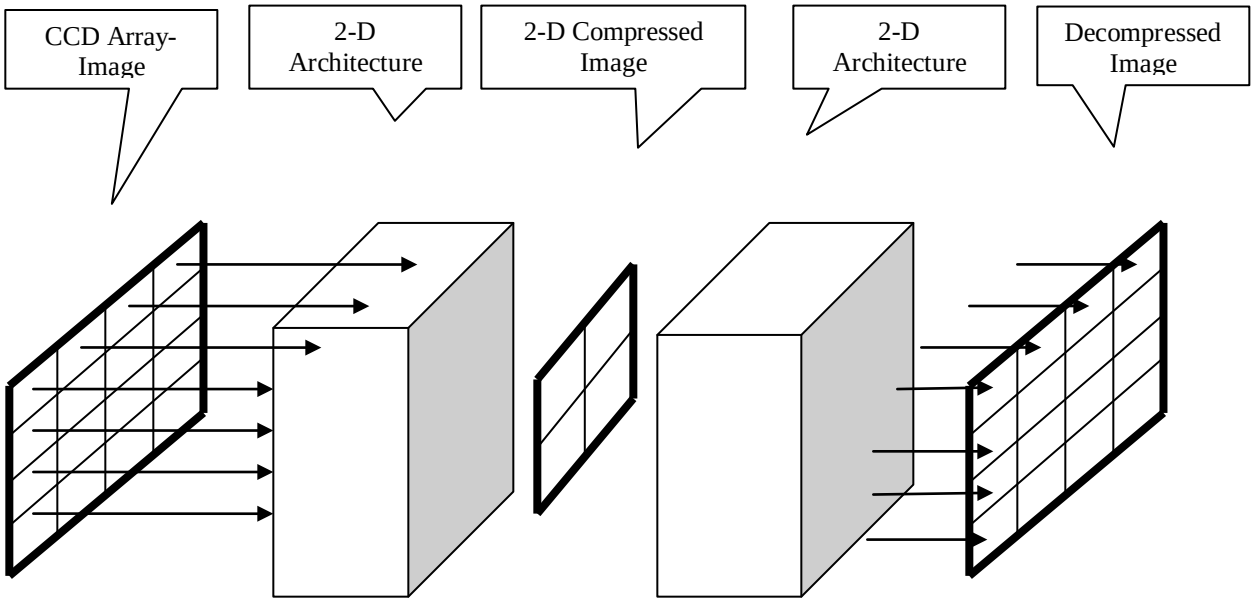


Figure 5.1 2-D multilayered neural network architecture

For every computation the input collected from the image sensor need to be multiplied by the weight matrix and bias elements, hence weight and bias elements need to be stored. The compressed data can be further compressed using suitable 2-D encoding techniques. Hence storage space is required to store the compressed data. At the

decompressor stage, storage space is required to store the weight, bias, compressed data and decompressed data. As this research focuses on only compression and decompression, data encoding and decoding of the compressed data is not within the scope of this work. The objective of this chapter is to design, model, simulate, implement and test an optimum neural network architecture optimizing area, power and speed. The speed, area and power performances are tested for 4 x 4 input size, the results obtained can be extended to image sizes larger than 4 x 4.

5.3 Neural Network Design and Implementation

As discussed in chapter 2, neural network architectures can be realized using digital circuits, analog circuits and hybrid circuits. The basic building blocks required for neuron implementation are multiplier, adder, and network function and storage elements. During initial stages of research work, analog implementation was found to be suitable for hardware implementation. To analyze performances of analog neuron, a simple multilayered neural network is considered. A 2:3:1 neuron is considered for analysis. The network architecture is shown in Fig. 5.2. The neuron has two input layers three hidden layers and one output layer. Two inputs V_1 and V_2 are connected to the neurons in the hidden layer through weights W_{11} to W_{16}. The outputs of the hidden layer are connected to the output layer through weights W_{21} to W_{23}. The final output is a_{21}.

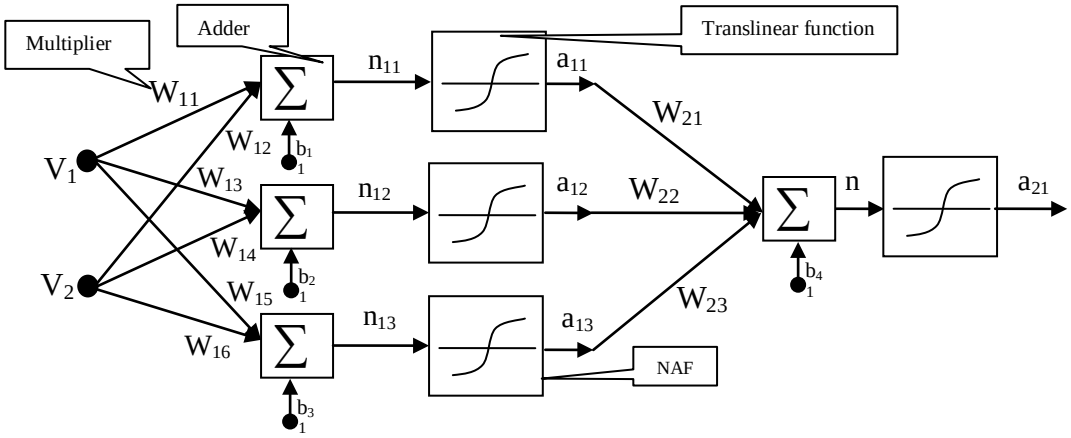


Figure 5.2 Block diagram for 2:3:1 neuron

5.4 Analog components for Neural network Architecture

The architecture shown in the Fig. 5.2 multiplies the input V_1 and V_2 with the weights W_{11} to W_{16} and the multiplied outputs are accumulated in the adder along with the bias b. The accumulated output n is then passed through the Neuron Activation Function (NAF) to produce the intermediate output a. The output layer processes the intermediate output and produces the final output. Analog blocks required to realize the network shown in Fig. 5.2 are:

1. Multiplication block (Gilbert cell)
2. Adders (with output of multiplier being current, currents from various multipliers are accumulated at a node, thus realizes a adder)
3. NAF block (nonlinear function)

5.4.1 Multiplier Design

The Gilbert cell is used as the multiplier block. The schematic of the Gilbert cell is as shown in the Fig. 5.3.

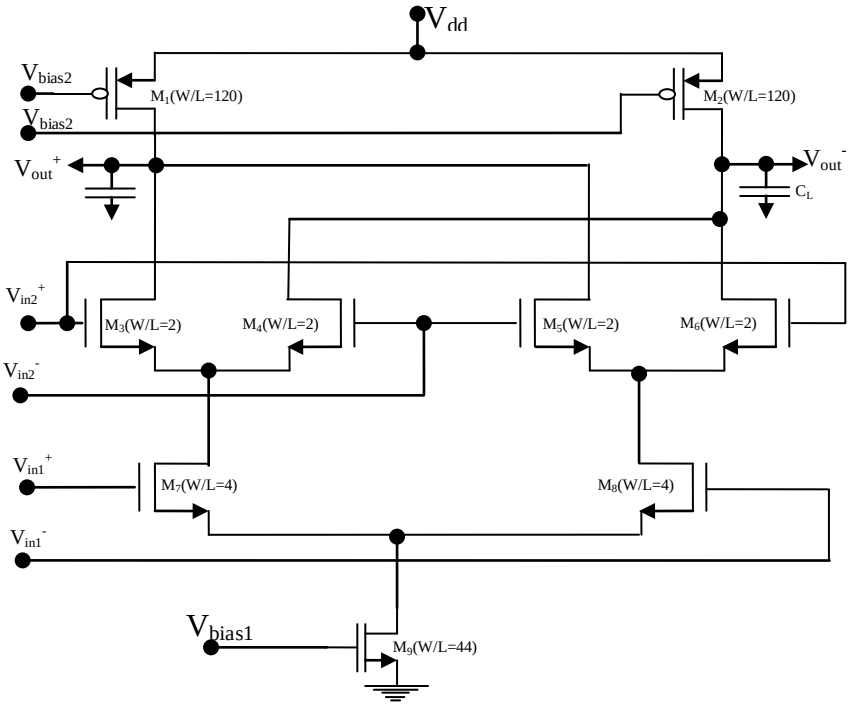


Figure 5.3 Gilbert cell multiplier

V_{in1} and V_{in2} are two analog inputs that are multiplied by the Gilbert cell, the output current flows through the output node $V_{out,}$ into the adder circuit, which is a current node. In order to verify the functionality of the Gilbert cell a load capacitance C_L is connected at the output node V_{out} to measure the output voltage. Thus the load capacitance converts current to voltage. Design of Gilbert cell multiplier is discussed in the next section. The Gilbert cell works in the sub threshold region. The current expression for NMOS transistor to work in the sub threshold region is given by the following equations (Razavi 2002, Eric A. Vittoz 2006):

$$I_{ds} = I_o e^{\frac{q[V_g - nV_s]}{nKT}} (1 - e^{-\frac{qV_{ds}}{nKT}}) \tag{5.1}$$

Where, all the voltages V_g, V_s and V_d are taken with respect to the bulk voltage V_b.

KT/q= 25mV at room temperature.

n = 1.2 to 1.6 slope factor. (5.2)

K- boltzmans constant, T – Temperature, q-electron charge

$$I_0 = 2n\mu C_{ox}\left(\frac{KT}{q}\right)^2 \frac{W}{L} \quad e^{-(\frac{V_{t0}}{nKT})q} \tag{5.3}$$

Cox- capacitance per unit area, µ-mobility of electron, W-width of transistor, L-length of transistor, Vto-threshold voltage

The current equation for PMOS is same as equation (5.1), but all the voltages have opposite signs so equation (5.1) will change to

$$I_{ds} = I_o e^{\frac{q[-V_g + nV_s]}{nKT}} (1 - e^{\frac{qV_{ds}}{nKT}}) \tag{5.4}$$

When $qV_{ds} / KT \geq 4$, or in other words V_{ds} is equal to 100mV, the term $e^{-qV_{ds}/KT}$ is approximately zero. Equation (5.1) becomes

$$I_{ds} = I_o e^{q[V_g - nV_s]/nKT} \tag{5.5}$$

Equation (5.5) is the saturation current in the sub-threshold region and I_{ds} is independent of V_{ds}.

5.4.2 Design of Multiplier Block

In the neural network architecture when the voltage specified to the Gilbert cell is 3.5 V (considering the worst case for non subthreshold operation of the MOS transistors for a Vdd of 5 V), the outputs should not exceed 3.5 V as it should be limited to less than the threshold voltage. The bias current for the Gilbert cell is assumed to be 2 µA (Ali Naderi et. al 2008, Burcu Kapanoglu and Tulay Yildirim 2003, Navin Saxena and James J. Clark 1994). Now considering each transistor in the saturation region one can design circuit for the Gilbert cell multiplier. For 0.35 micron technology, and calculating for geometries of transistor M9 (Fig. 5.3) with I_o=3.863 nA and VDS = 150 mV, using equation (5.1) $(W/L)_9 = 44.5$ for Vgs = 1 V. Now this current is divided into two paths one from M_7 and other from transistor M_8. I_{ds} for both the paths are 1 µA. Substituting this value in equation (5.5) while considering the output swing as 0.5 V, geometries for M_7 and M_8 are $(W/L)_{7\text{-}8} = 4$.

Further, the current through each branch is divided into two branches containing M_3, M_4 and M_5, M_6. The current through each branch is now 0.5 µA. considering transistor M_3, again same condition is seen as was for M_7, since both will have same maximum inputs in the threshold region, $(W/L)_{3,4,5,6} = 2$. For PMOS operating in sub threshold region, I_o = 2.66 nA. Current carried by each PMOS is 1 µA. Designing for this current using equation (5.4) $(W/L)_{1\text{-}2}$=120.7. The designed Gilbert cell is simulated using HSPICE.

The simulation result shown in the Fig. 5.4 is for the multiplication of two voltages V_{in1} and V_{in2}. V_{in1} is 0.2 V pp and 10 MHz frequency and V_{in2} is 0.2 V pp and 1 MHz frequency is applied as the input to the multiplier to test its performances. Fig. 5.4 shows the results for the multiplier, first and second signals are the inputs, third signal is the actual output, fourth is the theoretical output. The results show that the theoretical and

practical values match, but the signal strength of practical and theoretical results vary by a factor of 10 (Matlab results are compared with Spice).

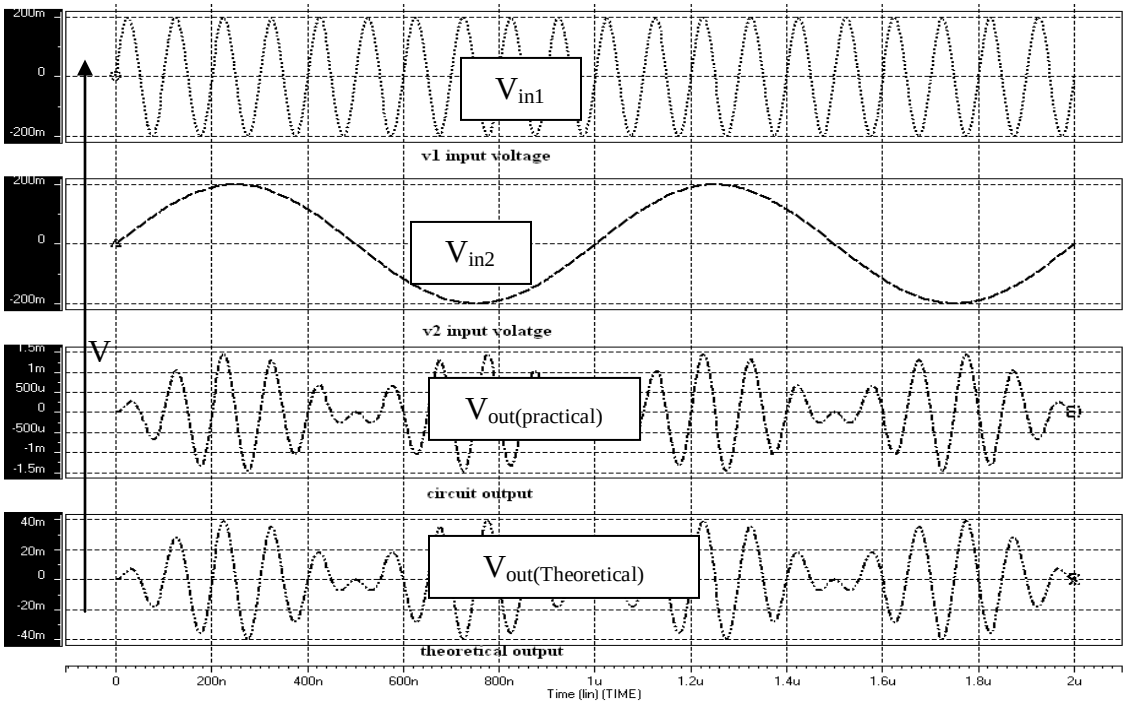


Figure 5.4 Gilbert cell multiplier results

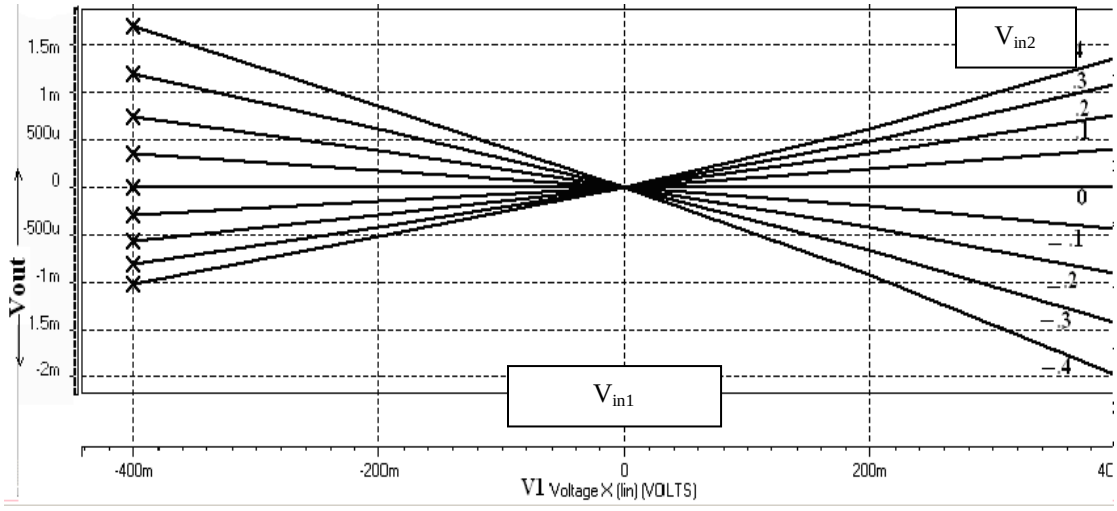


Figure 5.5 DC characteristics of Gilbert cell multiplier

Theoretical results are obtained by multiplying two input voltages, in Matlab and are imported to Spice for comparison with practical values. V_{out} is the multiplication of V_{in1} and V_{in2} voltage computed by the circuit. The output amplitude is 1.5 mV pp. The V_{out} obtained matches with the theoretical output. Fig. 5.5 shows the result for the DC characteristics for the Gilbert cell multiplier, which is a plot of V_{in1}* V_{in2}.

The input voltages V_{in1} and V_{in2} are varied from -0.4 V to 0.4 V. The characteristic shows a maximum of 2 mV output. For gray scale image having pixel values between 0-255, requires voltage values between -2 mV to +2 mV. Transistors M_7 and M_8 are matched, similarly transistors M_3 and M_6, M_4 and M_5 are matched thus minimizing the effect of temperature leading to non linear behaviour (Andreas G. Andreou *et al.* 1991). In this work, current is assumed to be 2 µA, based on the discussions provided in Razavi 2002. From the simulation results, it is observed that there is non-symmetrical response in the DC characteristics of Gilbert cell multiplier. In order to overcome these nonlinear effects in analog multiplier cell, hybrid multiplier is realized. Operating region of Gilbert cell multiplier is limited to +0.2 V to -0.2 V.

5.4.3 Adders

The output of the Gilbert cell is in the form of current (transconductance). The node connecting the respective outputs of the Gilbert cell, act as adder itself. The load capacitance (0.1 pF) connected at the output is used to convert output current to voltage at the output node.

5.4.4 Neuron Activation Function (NAF)

Neuron activation function designed here is Tansig. The circuit that can exhibit Tansig relation between output and input is the differential amplifier.

5.4.5 Differential Amplifier Design as a Neuron Activation Function

Differential amplifier when designed to work in the sub-threshold region acts as a neuron activation function. Consider a simple differential pair shown in the Fig. 5.6. Differential pair consists of five transistors, inputs differential V_1 and V_2 control the output current $I_{out.}$

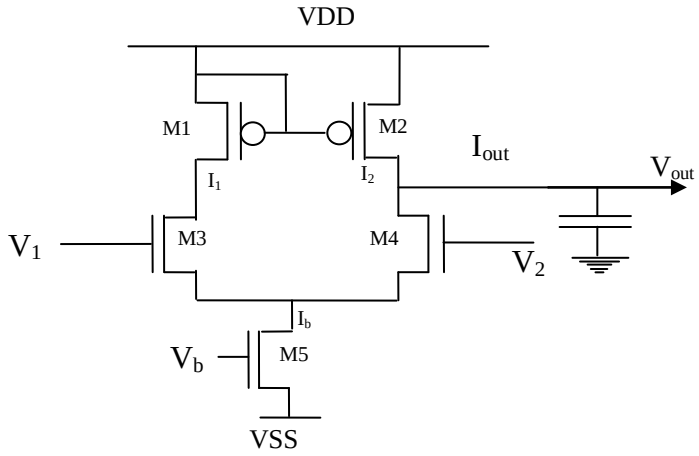


Figure 5.6 Simple differential amplifier

The currents in the sub-threshold region are given by equation (5.5) and assuming source and bulk to be shorted and both transistors have same W/L, the currents in M_1 and M_2 transistor is given by (Bose N. K., Liang P, 2002),

$$I_1 = KI_o e^{\frac{q[V_1]}{nKT}} \tag{5.6}$$

$$I_2 = KI_o e^{\frac{q[V_2]}{nKT}} \tag{5.7}$$

Also $I_1+I_2 = I_b$.

Where I_b is bias current for the differential amplifier. Combining equation (5.6) and (5.7)

$$I_b = KI_o e^{\frac{q[V_1]}{nKT}} + KI_o e^{\frac{q[V_2]}{nKT}} \tag{5.8}$$

Implying that

$$K = \frac{I_b}{I_o e^{\frac{q[V_1]}{nKT}} + I_o e^{\frac{q[V_2]}{nKT}}} \tag{5.9}$$

Now differentiated output current is defined as $I_1-I_2=I_{out}$ or in other words

$$I_{out} = I_b \frac{I_o e^{\frac{q[V_1]}{nKT}} - I_o e^{\frac{q[V_2]}{nKT}}}{I_o e^{\frac{q[V_1]}{nKT}} + I_o e^{\frac{q[V_2]}{nKT}}} \tag{5.10}$$

Trigonometric function tanh(x) is given by

$$\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \tag{5.11}$$

Thus equation (5.10) can be transformed to be in the form

$$I_{out} = I_b \frac{1-e^{2\frac{q[V_2-V_1]}{2nKT}}}{1+e^{-2\frac{q[V_2-V_1]}{2nKT}}} \tag{5.12}$$

Thus

$$I_{out} = I_b \tanh\left(\frac{q[V_2-V_1]}{2nKT}\right) \tag{5.13}$$

Equation (5.13) proves the functionality of the differential amplifier as a tan sigmoid function generator. As is evident from equation (5.13) I_{out} is the combination of bias current and the voltage input. Thus this can also be used as a multiplier when one input is current and the other is voltage. For the bias current I_b of 150 nA, M_5 transistor is first designed for gate voltage of 0.1 V. Thus using equation (5.5) $(W/L)_5$ value is calculated as 3.3. This current is divided into two branches with each branch carrying 75 nA of current. The maximum voltage for transistors M_3 and M_4 is 0.5V. The (W/L) value calculated for M_3 and M_4 is 2. The PMOS M_1 also carries the same current as M_3, gate voltage is considered for the 0.1 V (minimum voltage to keep it in saturation in sub threshold conduction region) the $(W/L)_{1\text{-}2}$ is calculated to be 1.5. The neuron activation function is basically a tanh(x) function. The theoretical value of y = tanh (x) is shown in Fig. 5.7.

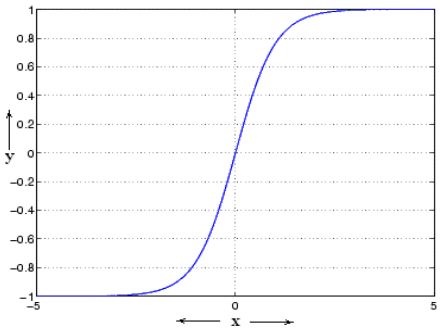


Figure 5.7 Graph showing y = tanh(x)

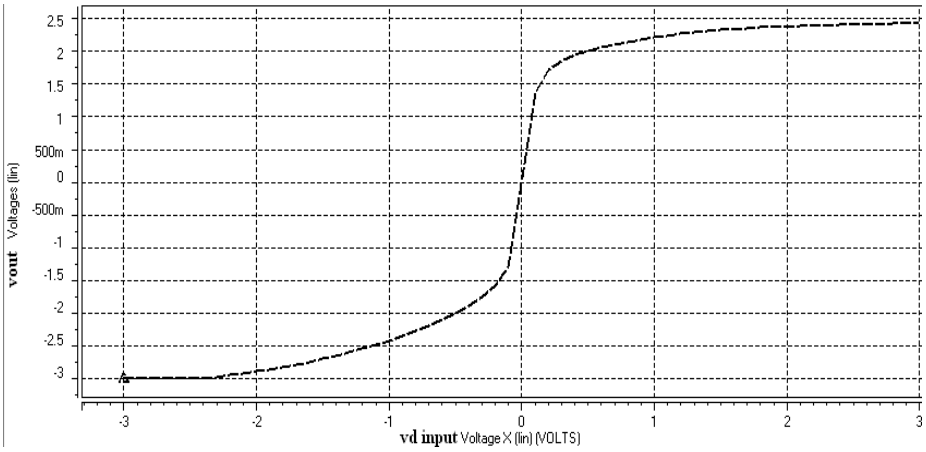


Figure 5.8 Circuit output for neuron activation function block (tan)

The simulation results of the designed nonlinear function are shown in Fig. 5.8. The results match with the theoretical values. To verify the results, the input is varied from -3 V to +3 V, the output is measured and it varies from +3 V to -3 V. The basic building blocks for the network are designed and are verified for its performance and functionality. The results show that the multiplier has output voltage swing of 3 mV peak to peak for an input range of 0.5 V. This ensures that the network performance should be limited to these voltage extremes. As the output of Gilbert cell multiplier is current, the adder circuit is a node. The current at a given node gets added as per Kirchhoff's current law. The activation function is either Purelin or Tansig; Purelin does not require any circuit. Tansig is realized using the neuron activation function. The Gilbert cell multiplier requires 6 transistors, activation function requires 5 transistors, and hence to realize a

neuron in analog domain, minimum of 11 transistors are required. Hence analog implementation requires less number of transistors compared to digital counterpart. Based on the building blocks analog neural network is realized.

5.5 Realization of Neural Architecture using Analog Components

The components designed are used to implement the neural network architecture. Fig. 5.9 shows the neural network architecture using analog components. This block is used as the neuron activation function as well as for the multiplication purpose. The *mult* is the Gilbert cell multiplier, the *fun* is the neuron activation function circuit.

The hidden layer is connected to the input layer by weights in the first layer named as w_{1i}. The output layer is connected to input layer through weights w_{2j}. The *op* is the output of 2:3:1 neural network.

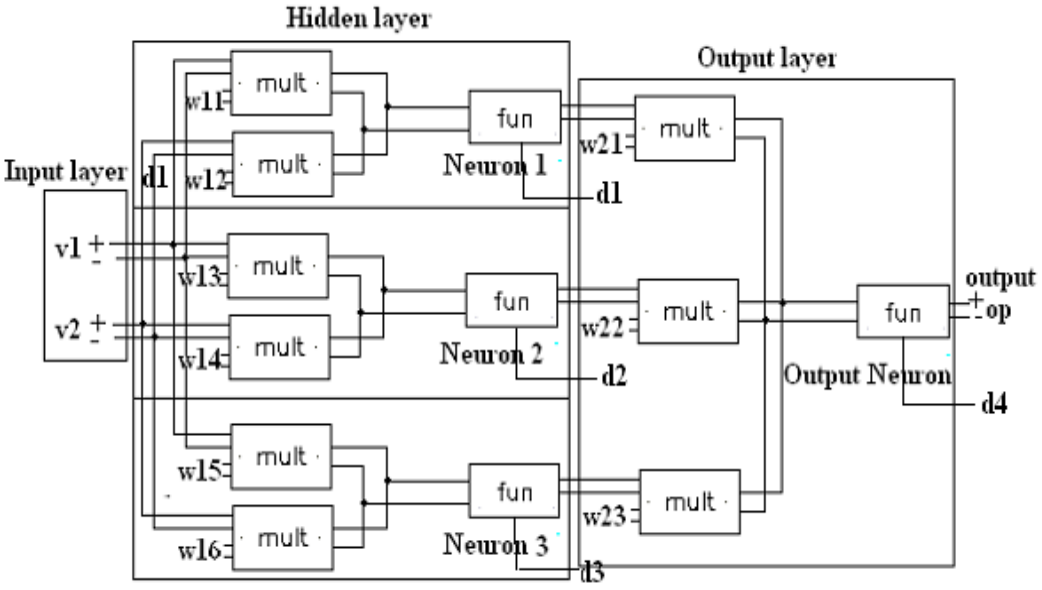


Figure 5.9 Implementation of the neural architecture using analog blocks

5.5.1 Backpropagation Algorithm

The training is an important part in the neural network architecture. Back propagation algorithm is used to train the network Chun Lu, Bing-xue Shi and Lu Chen (2002).

a) Updating the Output Layer Weights

In equation (5.12) I_{out} is the multiplication of the input applied to the differential amplifier and the bias current of the amplifier. On the same basis the differentiation current is multiplied to the target (d_i) and output (a_i) difference, implementing $\partial_i = (a_i - d_i)d(a_i)$. Next step is to calculate the weight update using the term $\Delta w_{ij} = \eta \partial_i a_i$. The obtained ∂, is then multiplied with the outputs of the hidden layer (input for the output layer) as shown in the Fig. 5.10. The output from the *mult* blocks is used as weight update for the weights in the output layer.

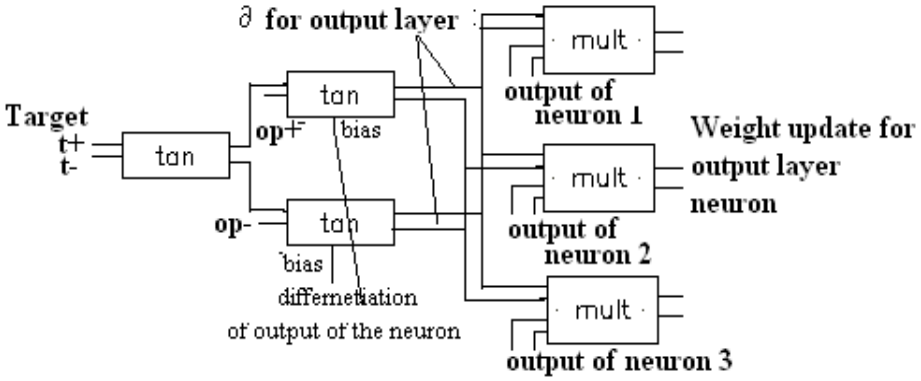


Figure 5.10 Block diagram for weight update scheme for the output neuron

b) Updating the Hidden Layer Weights

The hidden layer weights in the architecture are updated from the errors propagating from the output layer. This update requires the realization of $\partial_{hiddenlayer} = d(a_i)\sum w_{ij}\partial_i$, which deals with the ∂ formation for the hidden layer, ∂ is to be formed for each neuron in hidden layer. $\partial_{1,2,3}$ is formed considering the weight and ∂ output of neuron as shown in the Fig. 5.11. The output of the multiplication is then given to the differential amplifier with the bias current as the differentiation of the respective neuron output in the hidden layer. The ∂ formed is then used to update the weights in the hidden layer as implied by $\Delta w_{ij} = \eta \partial_i a_{input}$. The a_{input} is the input to the hidden layer, in this case the inputs v1 and v2. The network designed is trained using back propagation algorithm and is tested for both analog and digital functions.

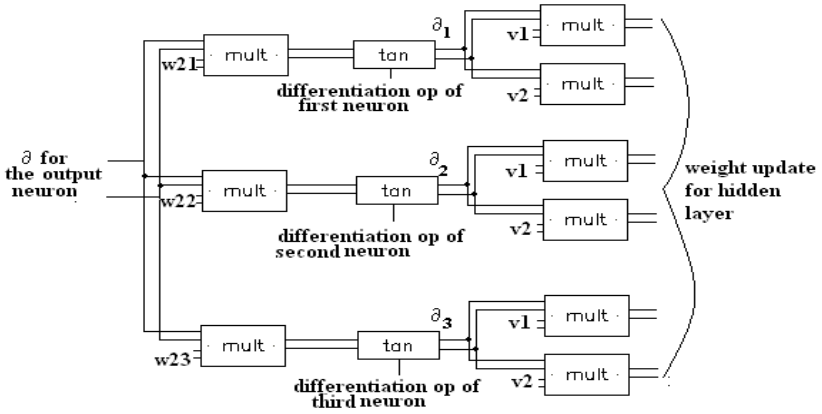


Figure 5.11 Block diagram for weight update scheme for hidden layer neuron

5.5.2 Validation for Digital Operation

The neural network architecture was verified for logic gates like AND, OR, XOR and NOT. Fig. 5.12 shows the AND operation learned by the 2:3:1 Neural Architecture. The input voltages v1 and v2 are given to the architecture along with the target. The input voltages swing form 1 V to –1 V. Target given to the circuit also varies from 1 V to –1 V. The output generated by the neuron, as shown in Fig. 5.12, clearly follows the target.

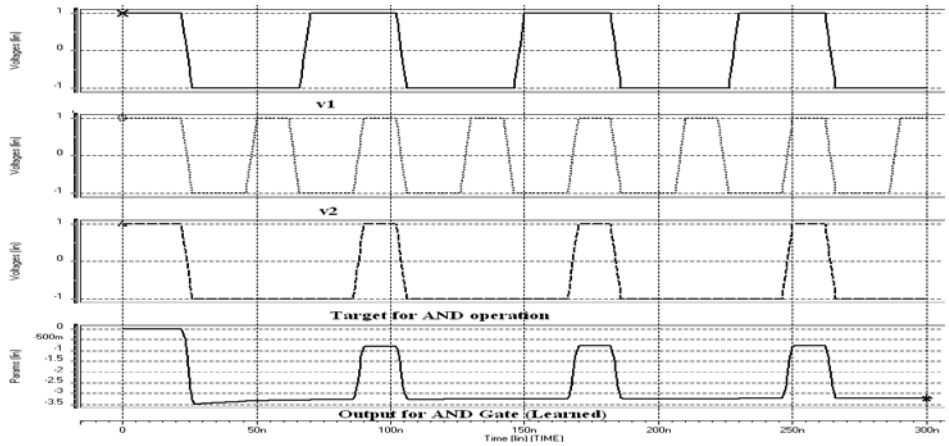


Figure 5.12 AND operation learned by 2:3:1 NN architecture

The output of the neural architecture swings from –0.726 V to –3.26 V (1.23 V swing). The weights are initialized to the value 1 V. Similarly the network was tested for other digital functions.

5.6 Image Compression and Decompression using NN Architecture

The neural network architecture is extended for the application of image compression and decompression. The simulation result for image compression and decompression are shown in the Fig. 5.13. The input v1 is a sine wave with 1 Vpp voltage and 5 MHz frequency, and v2 is a sine wave with 0.5 Vpp voltage and 10 MHz frequency. The compressed output is a DC signal of 233.63 nV. The decompressed output is shown in the same window Fig. 5.13. The decompressed output v1 is 1.2 Vpp with 5 MHz frequency and v2 is a 0.51 Vpp with 10 MHz frequency. As there is one output for 2 inputs there is a 50% compression.

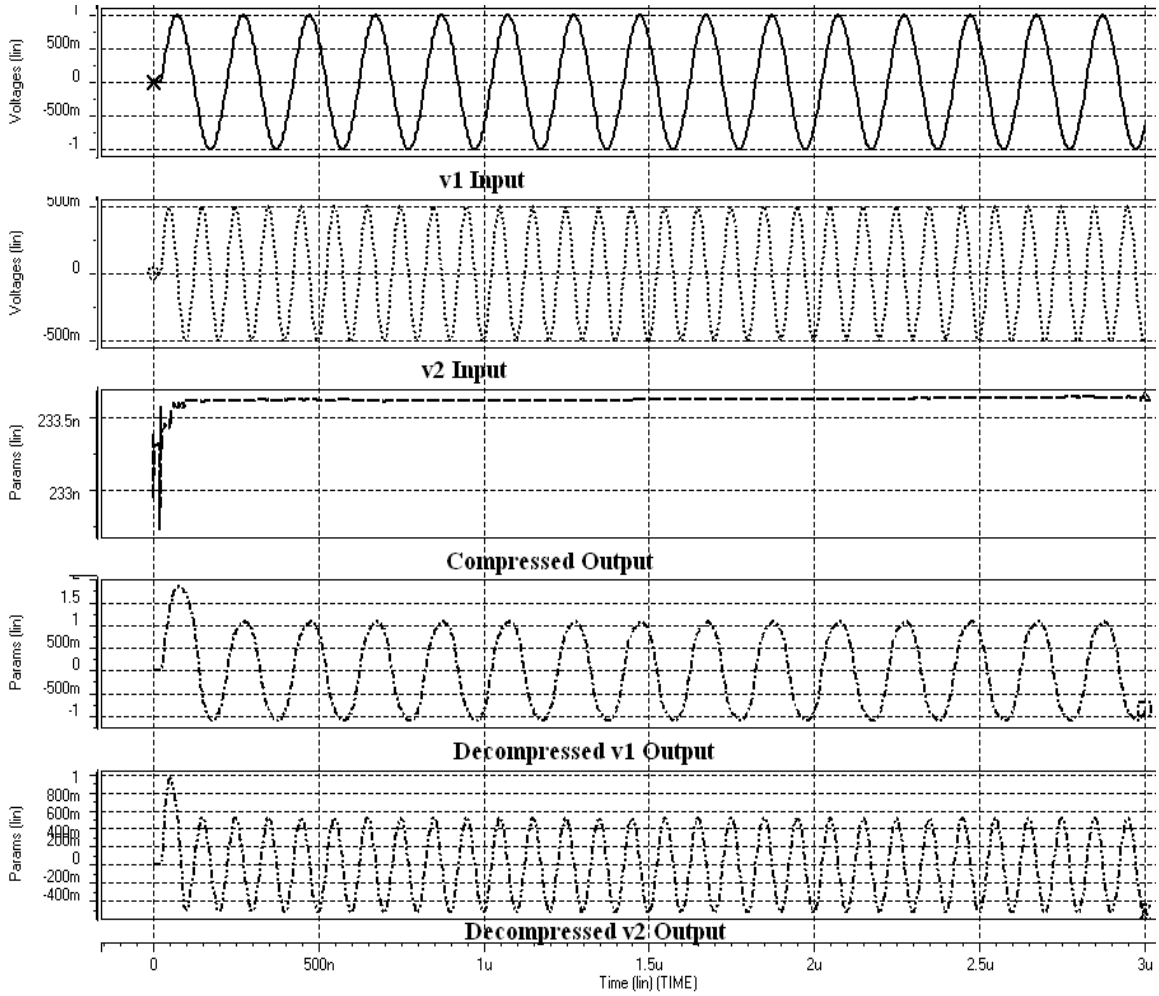


Figure 5.13 Image compression and decompression simulation

a) Limitations of the 2:3:1 neural network

Fig 5.14 shows the simulation results of XOR gate using the neuron designed. The network is trained to meet the functionality of XOR operation. The network is trained online, the training of the network automatically takes place once the inputs and outputs are presented. The trained network stores the weights using the parasitic capacitances in the neuron. The capacitances storing the weights are not capable of storing the weights for longer duration and hence, the functionality does not get realized as observed in Fig. 5.14. In Fig. 5.14, inputs applied to the network realizes XOR operation only for a short interval, if the inputs are held constant for some time, instead of realizing the required functionality, the network produces erroneous results. This is due to the fact that the parasitic capacitance discharges very quickly; hence to overcome this disadvantage, frequent training is required. This further adds to circuit complexity.

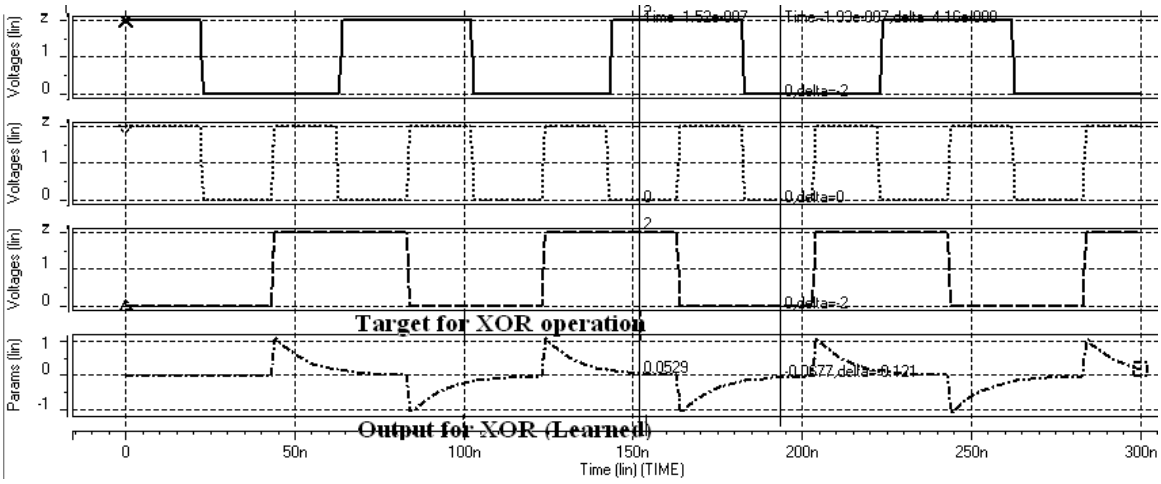


Figure 5.14 Limitation of 2:3:1 neuron for XOR operation

b) Weight storage and update mechanism

In order to make the network reliable, new circuit is proposed and is shown in Fig. 5.16. In this circuit, two capacitors are introduced into the circuit controlled by switches. The switches are controlled by two clocks clkI and clkW having complementary phase. The weights for the proposed neural architecture are stored on a capacitor. Fig. 5.15 shows the update mechanism and initialization of the weights.

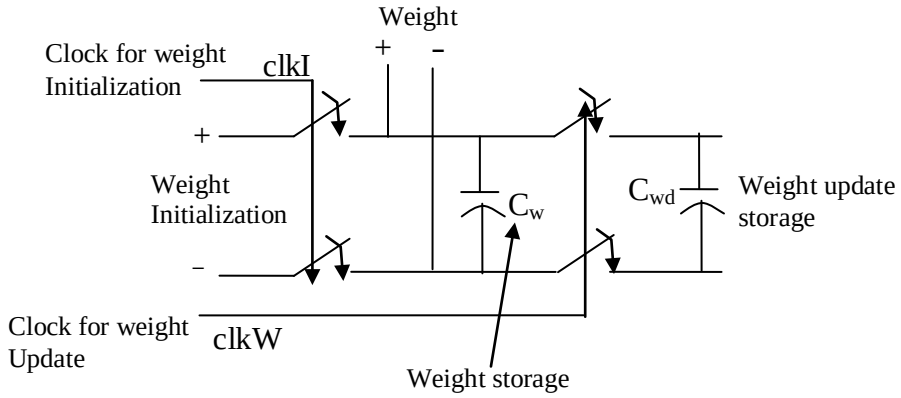


Figure 5.15 Weight update and initialization scheme

C_w is used to store the weight and C_{wd} is used to store the weight update. Clock signal clkW, is used for updating the weight. Whenever the clock is high the weight is updated, else there is no update and previous value of the weight is maintained. The weight initialization can also be done external to the chip, using clock ClkI. Whatever voltage is applied to weight initialization line, it is given to the C_w when clkI is high. One has to make clkI low before starting to train chip. This mechanism eliminates the limitations of the circuit designed earlier. The results of analog neural network architectures for image compression and decompression are presented in Cyril and Pinjare (2009). However, the complexity in this circuit is that, the network consumes power, due to the fact that the network continuously getting trained, whenever the weight values reduces below a threshold, the trained weights need to be transferred from the main capacitor. This not only adds to circuit complexity but also has power loss. In order to overcome these limitations and a new architecture is required for efficient hardware implementation. The weight and bias elements obtained after training are converted to its digital equivalent and stored in external memory. Input to the network is directly fed from image sensors and hence the inputs are analog, as the weights are binary. Multiplication of analog inputs with digital weights is required to be performed. Hence, there is a need for a circuit that can perform this functionality with minimum cost overheads, reduced circuit complexity and also should be optimized with respect to area, time and power

consumption. The next section discusses the 2-D multilayered architecture in detail, and new hybrid architecture is proposed for hardware realization.

5.6.1 2-D multilayered Neural Network Architecture Design and Implementation

In chapter 4, the 2-D multilayered neural network was trained to obtain the weight and bias elements. In order to overcome the storage problem of the weight matrix, offline training is chosen. The software reference model developed in the previous section, computes the weights for the network. These weights are converted to binary values and stored in memory. The weight matrix is converted to a 7-bit binary number and is shown in Table 5.1. The weight matrix for the hidden layer is given in column 1 and the binary equivalent of the weight matrix is given in column 2. Column 3 and column 4 gives the output weight matrix and its binary representation. As the size of input weight matrix is 4 x 16 (64), each weight being represented by 7 bit number, a memory of size 64 x 7 = 448 bit memory is required. Similarly on the receiver section, 448 bit memory is required to store the weight matrix. As there are three networks, the total number of weight matrix required at the transmitter side is 448 * 3 = 1344 bits. The number of bias elements is 4 and 16. The total number of bits required to represent the weight and bias elements at the transmitter is 448*3 + 28*3 = 1428 bits. Nonlinear functions are realized using translinear principle. A detailed discussion on network implementation is presented in the next chapter.

Table 5.1 Binary equivalents of weight matrix

Input weight	Equivalent binary(Input weight * 32)	Output weight	Equivalent binary
-0.1348	(-04) 1000100	-25.9764	(-25) 1011001
0.0718	(+02) 0000010	-3.2596	(-03) 1000011
-0.3272	(-10) 1001010	+4.3240	(+04) 0001100
-0.3331	(-10) 1001010	-59.6807	(-59) 1111011
-0.4078	(-12) 1001100	-36.7612	(-36) 1100100

+0.1711	(+05) 0000101	-0.2085	(-00) 0000000
-0.0401	(-01) 1000001	+18.2802	(+18) 0010010
-0.2914	(-09) 1001001	-58.9398	(-09) 1001001
+0.5041	(+16) 0010000	-55.8463	(-55) 1110111
+0.2509	(+08) 0001000	-1.6886	(-01) 1000001
-0.1108	(-03) 1000011	+23.4713	(+23) 0010111
-0.2488	(-07) 1000111	-47.1475	(-47) 1101111
+0.3174	(+09) 0001001	-50.6820	(-50) 1110010
-0.7168	(-22) 1010110	-16.0312	(-16) 1010000

The input matrix captured from image sensors is analog in nature, for software reference model, the digital equivalents of analog intensities have been considered for training and testing the performances of the network. The adaptive 2-D network is trained offline. The weight matrix and bias elements calculated during training in digital. Hence can be easily stored using digital memory. As the input is analog, and weight is digital, there is a need for a hybrid architecture to multiply analog input with digital weights.

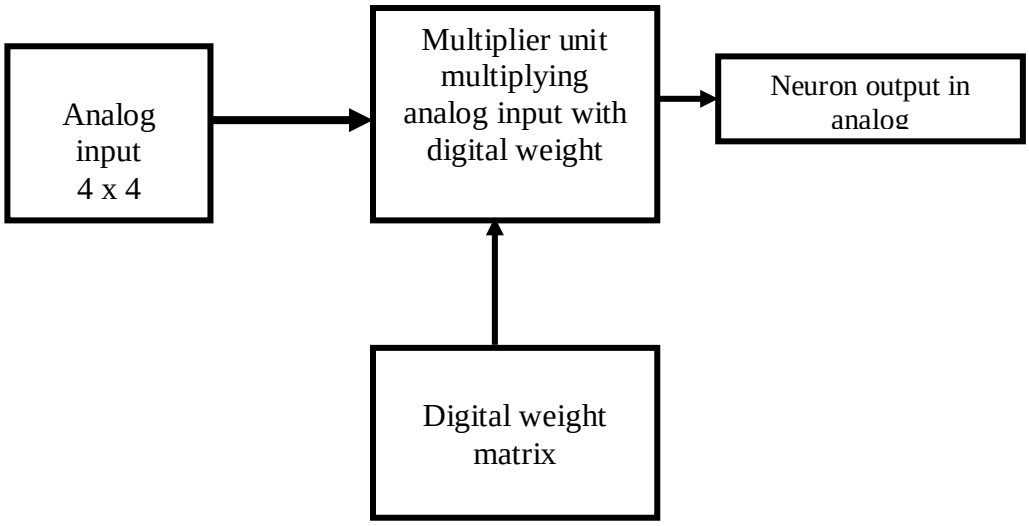


Figure 5.16 Hybrid neuron model for image compression

Hybrid architecture is proposed for implementation of 2-D multilayered neural network architecture. Figure 5.16 shows the top level block diagram of the proposed work. The major focus of this research work is the design and development of this hybrid architecture for image compression. Array of 4 x 4 analog intensities are multiplied by 4 x 16 digital weight matrixes to produce 2 x 2 analog outputs. Since the input is in spatial domain, output is also in spatial domain. This reduces the computational time and also leads to a new feature of 2-D coding technique. Table 5.1 shows the weight matrix represented in digital form. The total number of bits required to be stored in external memory representing the weights for the 2-D adaptive network (16:4:16) is calculated below:

As there are three networks each having 128 weights at the hidden layer and output layer, total number of bits per network is 896 bits (7 bits per weight, 128 weights * 5 bits). Total number of bits for three layers is 2688 bits. As there are 20 biases, each requires 7 bits hence the total number of bits to be stored are 2828 bits. Instead of using 7 bits for representing weights and biases, for reducing the circuit complexity of the network architecture 5 bits are used to represent the weights and biases. This introduces loss in data representation, the amount of error or loss occurring due to reduction in bit width is discussed in next section.

At the hidden layer digital representation of the weight matrix is multiplied with the 2-D input image intensities (analog) that are captured using the image sensors. The top level architecture of the 2-D multilayered neural network architecture is shown in Fig. 5.17. 256 x 256 image sub-divided into 4 x 4 block size is considered every time interval for compression and decompression. Input pixels $I_{1,1}$ to $I_{4,4}$ are first multiplied by the weight matrix and then added to from $Z_{1,1}$ to $Z_{2,2}$ compressed output. The weight matrix is digital, input and output is analog. Hence a hybrid circuit is required to multiply analog inputs with digital weights.

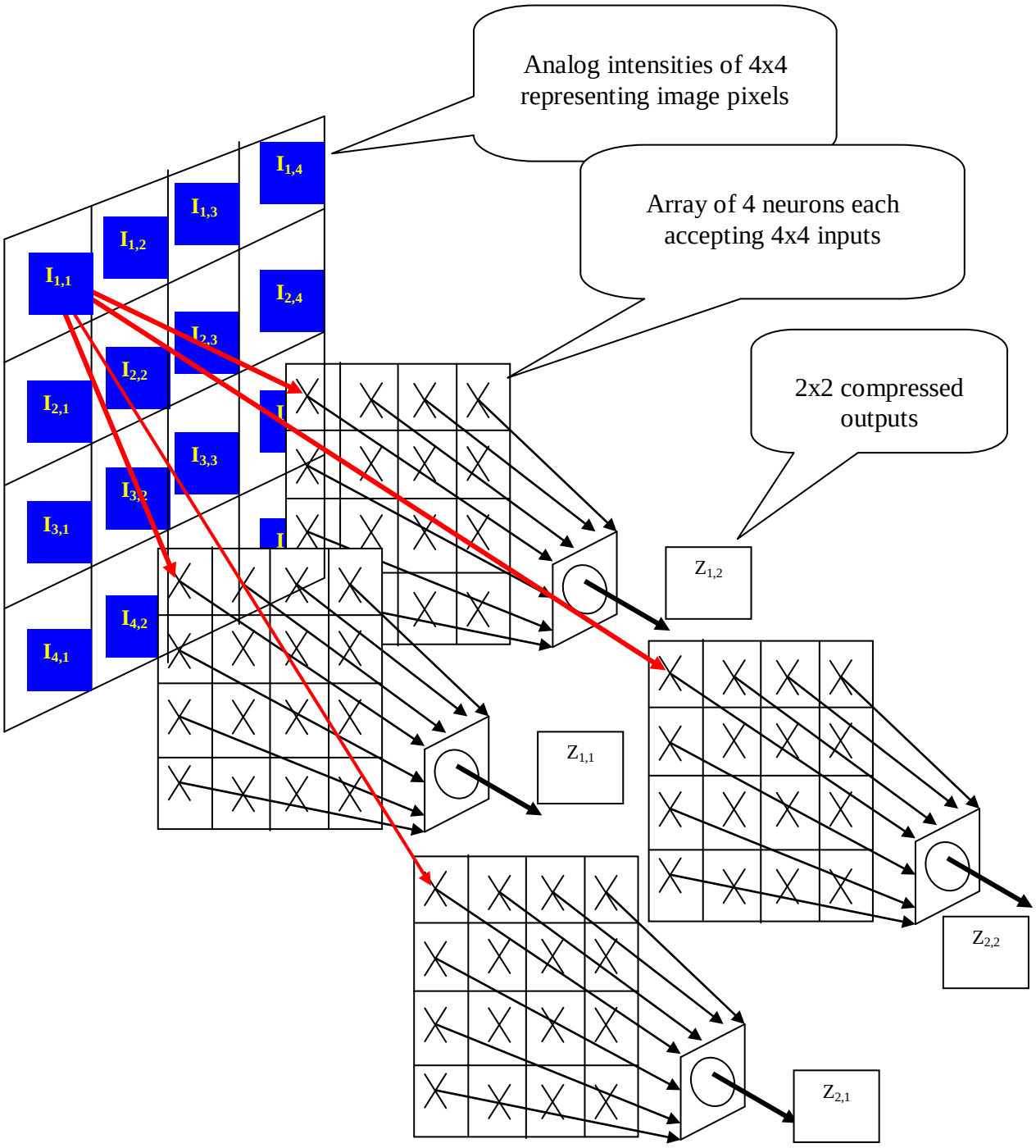


Figure 5.17 2-D architecture of neural network (Hidden Layer)

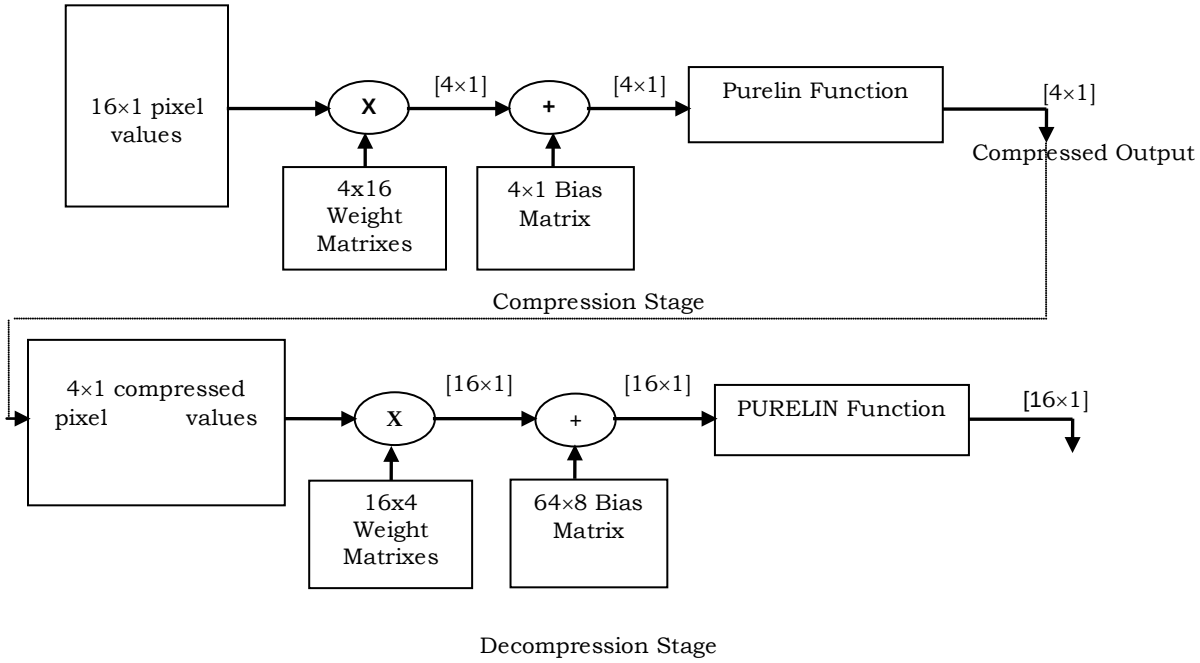


Figure 5.18 Block diagram of single neuron with input and hidden layer

$$\begin{bmatrix} W_{11} & W_{21} & W_{31} & \cdots & W_{161} \\ W_{12} & W_{22} & W_{32} & \cdots & W_{162} \\ W_{13} & W_{23} & W_{33} & \cdots & W_{163} \\ W_{14} & W_{24} & W_{34} & \cdots & W_{164} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ \cdot \\ \cdot \\ I_{16} \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \end{bmatrix}$$

$$\begin{bmatrix} W_{11} & W_{21} & W_{31} & W_{41} \\ W_{12} & W_{22} & W_{32} & W_{42} \\ W_{13} & W_{23} & W_{33} & W_{43} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ W_{116} & W_{216} & W_{316} & W_{416} \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ X_{16} \end{bmatrix}$$

Figure 5.19 Mathematical operation performed by the neural network

Fig. 5.19 shows the mathematical representation of the proposed architecture. As shown in Fig. 5.18, the network performs the mathematical operation as shown in Fig. 5.19. The multiplier stage is hybrid, adder and transfer function stage is analog, hence the name hybrid network architecture.

5.6.2 Hybrid Neural Network Architecture

The proposed design requires a circuit that can multiply analog inputs with digital weights. This is realized using the data converters. Data converters are most important building blocks of analog and mixed signal circuits for data conversion and communication applications as well. Digital to Analog also known as DAC'S, convert a digital or discrete signal into an analog signal or continuous timing signal, there are different types of digital to analog converters (Tiilikainen 2001, Wang, Fukatsu, and Watanabe 2001, Bugeja and Song 2000, Chi-Hung and Klaas 1998, Jacob and Nianxiong 1999 and Hyun-Ho and Cheong-Yong 2004).

a) Ideal Digital to Analog Converters

Digital to Analog converters convert digital signals into analog signal. The ideal block diagram of a DAC is shown in Fig. 5.20.

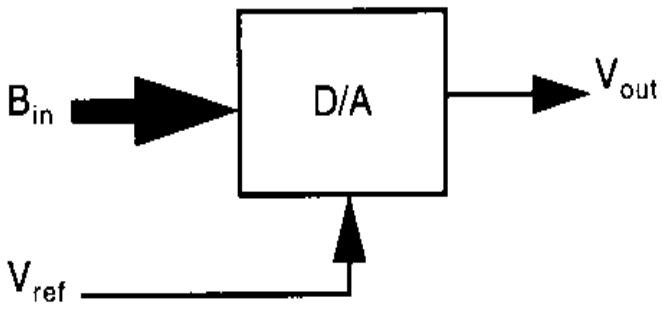


Figure 5.20 DAC ports

The input–output transfer curve for an ideal digital to analog data converter is shown in Fig. 5.21. Here, Bin is defined to be an N-bit digital signal (Wang, Fukatsu, and Watanabe 2001). Such that,

$$B_{in} = b_1 2^{-1} + b_2 2^{-2} + \ldots\ldots + b_N 2^{-N} \qquad (5.14)$$

Where, b_1 is the most significant bit and b_N is least significant bit. V_{out} represents the output of the digital to analog converter.

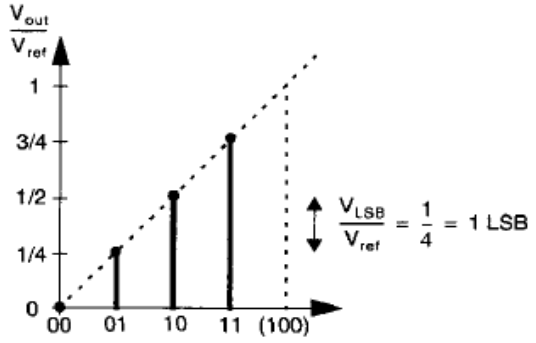


Figure 5.21 Input-output transfer curve for ideal 2-bit DAC (Tiilikainen 2001)

The relation between output voltage and reference voltage is given by equation (5.15),

$$V_{out} = V_{ref} (b_1.2^{-1} + b_2.2^{-2} + \ldots\ldots + b_N.2^{-N}) = V_{ref} B_{in} \qquad (5.15)$$

It is useful to define the value of V_{LSB} to be the voltage change when one LSB changes or mathematically is given by,

$$V_{LSB} = V_{ref} / 2^N \qquad (5.16)$$

Equation (5.15) relates V_{ref}, B_{in} and V_{out}. This forms the basis for the hybrid neural network architecture. The requirement is that the analog inputs are to be multiplied by digital weights; hence equation (5.15) can be modified to work as hybrid multiplier. Every charge coupled device produces an analog intensity equivalent to the image pixel. If this is taken as reference voltage Vref, this can be multiplied by the binary inputs Bin as per equation (5.15). The multiplier stage is realized using DAC architecture. The output of DAC is analog. Multiple partial products computed by the DAC, need to be added, hence an analog adder circuit is required. The output of adder should flow through the transfer function. If the network is linear, transfer function can be avoided, else the neuron activation function discussed earlier is used to realize the Tansig function. Output of each neuron is analog. This compressed output is also analog. In the output layer, the analog compressed samples are decompressed to analog output with the help of digital weights. Since DAC logic is identified to work as hybrid multiplier, there are multiple DAC architectures, hence a through literature review is conducted on various DAC architectures. Based on the literature review and design specifications, suitable DAC

architecture is selected, designed and verified for its performance. DAC is used as hybrid multiplier, it is important to know the limitations of the converter and how they affect the performance of the entire system. The outputs of image sensors are usually current, which get converted into voltage. Since the input pixels are current intensities, only current reference DAC are considered. A detailed study of different current steering DAC architectures is required to find the suitable architecture.

5.7 Current Steering DAC

The general architecture of a binary weighted current-steering DAC (Wang and Wey 1998, Anne *et al.* 1998) is shown in Fig. 5.22.

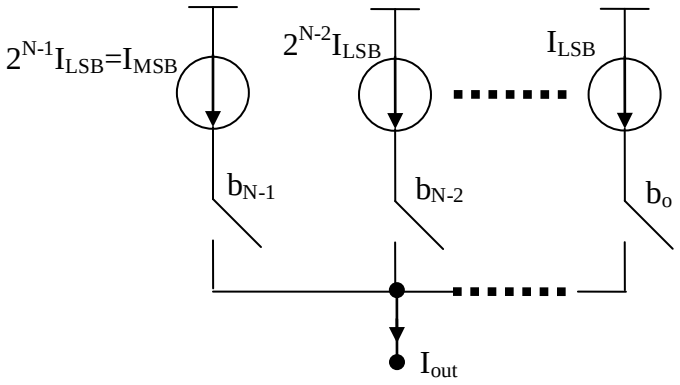


Figure 5.22 Binary weighted current string DAC

The switches are controlled by the input bits $b_0 - b_{N-1}$, where, N is the number of bits. The output current, of the DAC shown in Fig. 5.22 is given by,

$$I_{out}(K) = 2^0 I_{LSB}.b_0 + 2^1 I_{LSB}.b_1 + \ldots\ldots + 2^{N-1} I_{LSB}.b_{N-1} = I_{LSB}.K \qquad (5.17)$$

Where K is the digital input is given by,

$$K = 2^0 b_0 + 2^1 b_1 + \ldots\ldots 2^{N-1} b_{N-1} \qquad (5.18)$$

Depending upon the input bits, the switches are set to ON/OFF condition, that allows the current flow and at the node all the currents are added to give the analog output. The analog output of the circuit depends on the status of the binary switches. DAC circuits are used as hybrid multipliers in this work. The work proposed by Ryan *et*

al. (2004) is a multiplying DAC. The MDAC architecture is modeled using Spice and the results are tabulated in Table 5.2.

Table 5.2 Theoretical, practical outputs with the error of the reference architecture

Digital Inputs S0,S1,S2,S3	Theoretical Outputs in Amps	Practical Outputs in Amps	Error
0000	0	28e-15	28e-15
0001	3µ	2.0µ	1.0
0010	7µ	5.2µ	1.8
0011	11µ	7.2µ	3.8
0100	15µ	13.0µ	2.0
0101	18µ	15.5µ	2.5
0110	20µ	18.7µ	1.3
0111	22µ	20.7µ	1.3
1000	30µ	29.2µ	0.8
1001	33µ	30.7µ	2.3
1010	37µ	33.01µ	3.9
1011	41µ	34.5µ	6.5
1100	45µ	38.3µ	6.7
1101	48µ	39.4µ	8.6
1110	52µ	41.7µ	10.3
1111	56µ	61.0µ	5.0

The circuit is modeled using Spice, simulated using HSpice using 180nm technology. V_{ref} is set to 1.8 V. Out of 5-bit input, MSB bit is used for sign representation. The direction of current through the multiplier defines the sign of the weight matrix and hence the Table 5.2 shows the multiplication operation for 4-bit only, assuming all the weight inputs being positive.

Table 5.2 shows the output currents for different values of digital inputs ranging from 0000 to 1111 where S0 is the LSB and S3 is the MSB. The MDAC is working as a hybrid multiplier circuit. The digital inputs of the DAC are multiplied by the Current

references. The analog output of the DAC is the multiplied output in terms of current. In Table 5.2, input data 0001 multiplied by the corresponding current reference gives an output of 3 µA. The practical value obtained is 2 µA. The errors are calculated and are also tabulated. The results show that the maximum current is 61 µA and minimum current is 2 µA. From the result obtained the difference in output between expected and obtained is 10 µA maximum. This clearly indicates that the circuit has limited accuracy. In order to overcome the limitations, a new MDAC architecture is designed to meet the accuracy. The next section discusses the novel MDAC architecture.

5.7.1 Novel Hybrid Current Steering DAC Multiplier

In this section, a modified MDAC synapse is designed and demonstrated for different values of input current and the digital weights. The hybrid multiplier is designed and checked for its performance. The multiplier architecture is optimized for power, area and results are reported.

5.7.2 Proposed MDAC Architecture

The proposed MDAC architecture has a NMOS transistors connected in the form of R – βR, whereas the referred architecture consists of PMOS current mirror transistors connected, wherein a control is given to the circuit to either pass the outputs directly or through the current mirror circuit, The proposed MDAC has a better accuracy than the referred architecture. It has two main blocks:

1) Weighted Current Steering Circuit
2) MDAC Architecture

The current mirror used here is the folded cascoded current mirror. The main purpose of this current mirror here is to replicate the reference current irrespective of the load. The designed weighted current steering circuit produces current of 32 µA for an input reference current of 16 µA. This is achieved by doubling the widths of the successive transistors. This is based on the current equation of the MOS transistors in saturation region which states current is directly proportional to the width. This current is given as the input for the MDAC. Because of the heavy short-channel effect on a simple current

mirror, the weighted current steering circuit is realized with the cascoded configuration that can provide an accurate current with higher output impedance. The circuit as shown in Fig. 5.23 receives the input current I_{REF} and produces three output currents I_{in1}, I_{in2}, and I_{in3} that will be injected into the R-βR ladder networks. In this design, I_{in1}, I_{in2}, and I_{in3} are designed to be 16 (2^4) μA, 128 (2^7) μA, and 512 (2^9) μA, respectively. These values are selected as an example representing the input pixel samples.

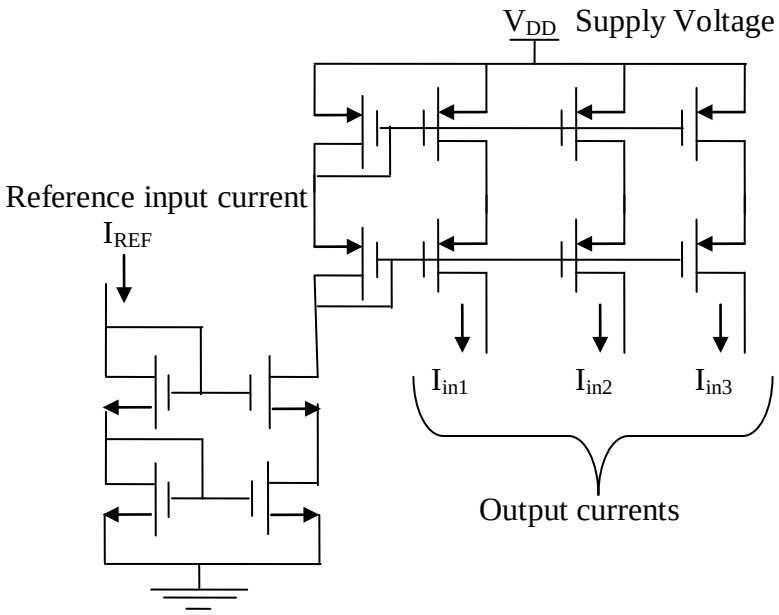


Figure 5.23 Weighted current steering circuit

The schematic of the current steering network for the proposed MDAC architecture is as shown in the Fig. 5.24. The input voltages required is at 1.8 V, the current source can be replaced by a transistor of the required width calculating the input and output characteristics to determine the region of the transistor which produces the required current at the output. The calculated voltage is given as the gate to source voltage. Fig. 5.24 shows the schematic of the circuit realized using Cadence Virtuoso. The design is carried out targeting 180 nm technology and the schematic capture of the

MDAC circuit matches with the theoretical circuit designed. Fig. 5.24 is circuit schematic of Fig. 5.23 captured using Virtuoso Spectre for simulation.

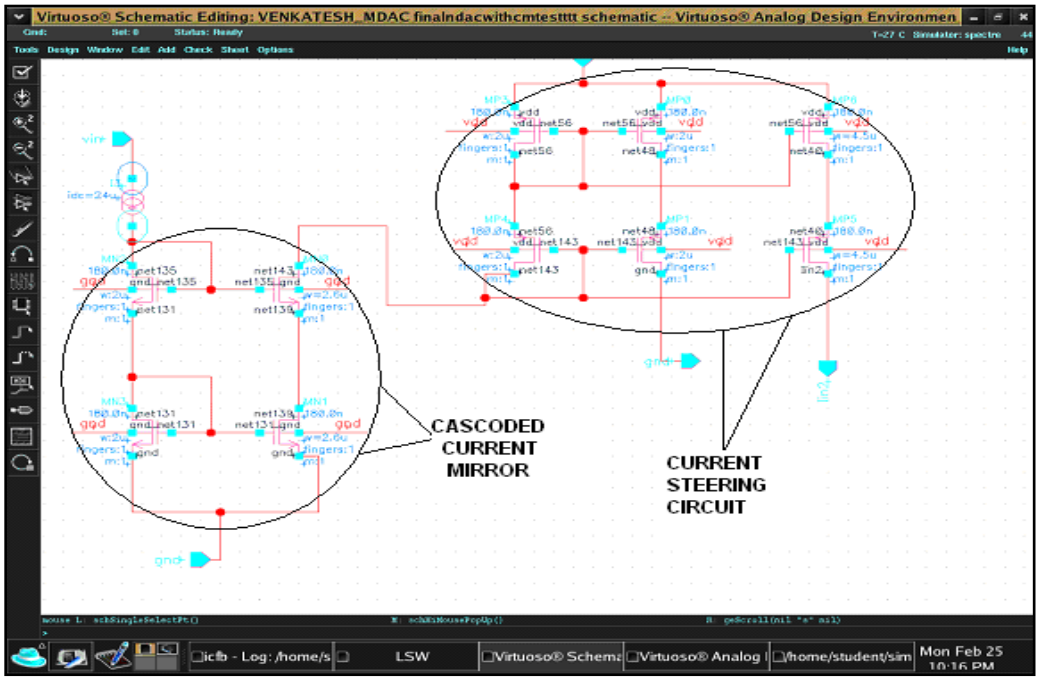


Figure 5.24 Weighted current steering circuit schematic from Virtuoso

5.7.3 MDAC Architecture - R-ßR Ladder Network

The R-ßR ladder network is modified from the R-2R ladder network such that a more accurate current-divider circuit can be obtained under the following condition as

$$\beta > 2\frac{1+\varepsilon}{(1-\varepsilon)^2} \tag{5.19}$$

Where ε denotes the error on the resistance resulted from the effect of device mismatch. For a TSMC 0.18 µm technology, the error range is

$$5\% \le \varepsilon \le 5\% \tag{5.20}$$

Substituting **ε** = 5% into Equation 5.19, β **is 2.3.** Fig. 5.25 shows the transistor-level R-βR ladder network. Each resistor in the R-βR ladder network is implemented by an NMOS transistor biased in triode region.

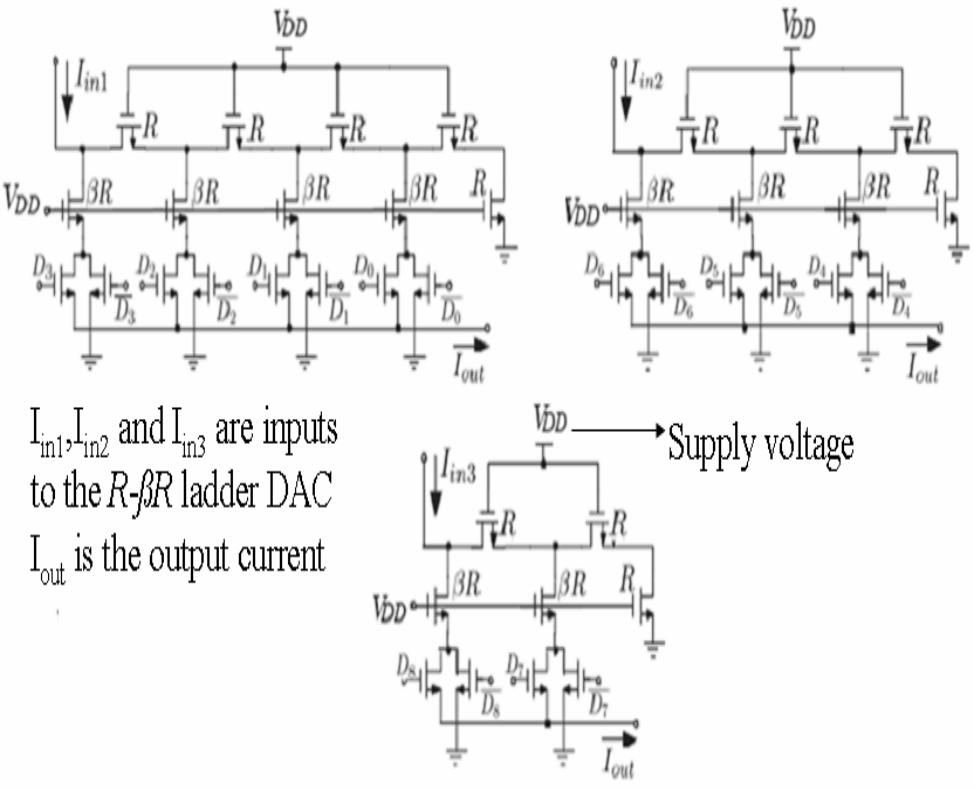


Figure 5.25 R- βR ladder network

An NMOS transistor operated at triode region can be characterized by equation (5.21)

$$I_D = \mu_n C_{ox} \frac{W}{L}\left[(V_{GS} - V_{Tn})V_{DS} - \frac{V_{DS}^2}{2}\right] \qquad (5.21)$$

The channel resistance R_n of an NMOS transistor biased in triode region can be derived as shown in equation (5.22)

$$R_n = \left(\frac{\partial I_D}{\partial V_{DS}}\right)^{-1} = \frac{1}{\mu_n C_{ox}\frac{W}{L}(V_{GS} - V_{Tn})}\alpha\frac{L}{W} \qquad (5.22)$$

Apparently, the channel resistance is inversely proportional to the aspect ratio of the transistor. Therefore, the R-βR ladder network can be easily implemented by specifying the aspect ratios of the transistors satisfying the relationship as shown in equation (5.23)

$$\left(\frac{W}{L}\right)_R = 2.3\left(\frac{W}{L}\right)_{\beta R} \tag{5.23}$$

Where $(W/L)_R$ and $(W/L)_{\beta R}$ represent the aspect ratios of the NMOS transistors implementing the resistances R and βR, respectively.

The transistors which form the R- βR structure have widths of 400 nm, 520 nm. The voltage controlled current switch has width of 400 nm each. The current mirror used has two top PMOS transistors of widths 2 µm each and bottom 2 transistors have widths of 2.8 µm each. The length used is 180nm. The values of W/L for transistors used in MDAC circuit are tabulated in Table 5.3.

Table 5.3 MDAC Transistors widths Tabulation for Proposed Architecture

Transistors	Widths for 180nm technology with length equals 180nm
MN1 to MN4	400.0 nm
MN5 to MN9	520.0 nm
M10 to M18	400.0 nm
MP19 to MP20	2.0 µm
MP21 to MP22	2.8 µm

MDAC architecture proposed by (Ryan Kier, J., Reid Harrison, R. and Randall Beer, D., 2004) is modified to design new MDAC architecture. From the size of the widths the total comes to 920 nm which is the R-βR realization of the circuit. The resolution of referred architecture and proposed architecture is 4-bit, 1-bit for sign representation. The offset error of MDAC synapse proposed by Anne et al. (2001) is $28e10^{-15}$ and offset error for proposed MDAC synapse is $2.4e10^{-12.}$ The error gain for the MDAC synapse proposed by Anne et al. (2001) is 4 LSB and gain error for the MDAC synapse proposed is 0.5 LSB.

The proposed architecture has better results compared with the reference model. The error between the theoretical and practical is reduced from 10 µA to 0.9 µA. The voltage swing of the proposed network is less compared to the reference model. However, the error between the theoretical and simulation values is maximum of +/- 0.9. Hence this circuit has good accuracy.

Table 5.4 gives the table of the different values of the output currents for the different digital inputs which are known as the weights, the digital weights is changed from 0000 to 1111 for this 4 bit architecture, the D_0 is the MSB and the D_3 is the LSB, the differential error is also calculated and tabulated in the Table 5.4.

Table 5.4 Comparisons of theoretical, simulation outputs for proposed architecture

Digital Inputs D0,D1,D2,D3	Theoretical Outputs in Amps	Simulation Outputs in Amps	Error
0000	0.00	2.3e-12	2.3e-12
0001	01.30µ	01.7 µ	0.4
0010	03.02µ	03.6 µ	0.5
0011	04.30µ	05.2 µ	0.9
0100	06.90µ	07.5 µ	0.6
0101	08.25µ	08.7 µ	0.4
0110	09.96µ	10.5 µ	0.5
0111	11.30µ	12.1 µ	0.8
1000	16.00µ	16.0 µ	0
1001	17.30µ	16.7 µ	0.6
1010	19.02µ	18.8µ	0.1
1011	20.30µ	20.0µ	0.2
1100	22.90µ	23.0µ	0.1
1101	23.80µ	23.9µ	0.1
1110	25.90µ	26.8µ	0.9
1111	28.20µ	28.6µ	0.4

The layout for the proposed MDAC architecture is captured using Cadence Virtuoso and the post layout simulations are carried out to analyze the area, timing and power performances.

5.8 Layouts of Proposed MDAC

MDAC layout is designed using Virtuoso tool. The design is developed using 0.18µ technology. The MDAC layout is shown as in Fig. 5.26. After designing layout first step is design rule check. When there are no DRC errors the design is checked for Layout versus Schematic check i.e. the layout is verified with respect to the schematic whether the connections made in the layout are same and correct compared to schematic. For the schematic shown in Fig. 5.25, layout is drawn using Cadence Virtuoso is shown in Fig. 5.26.

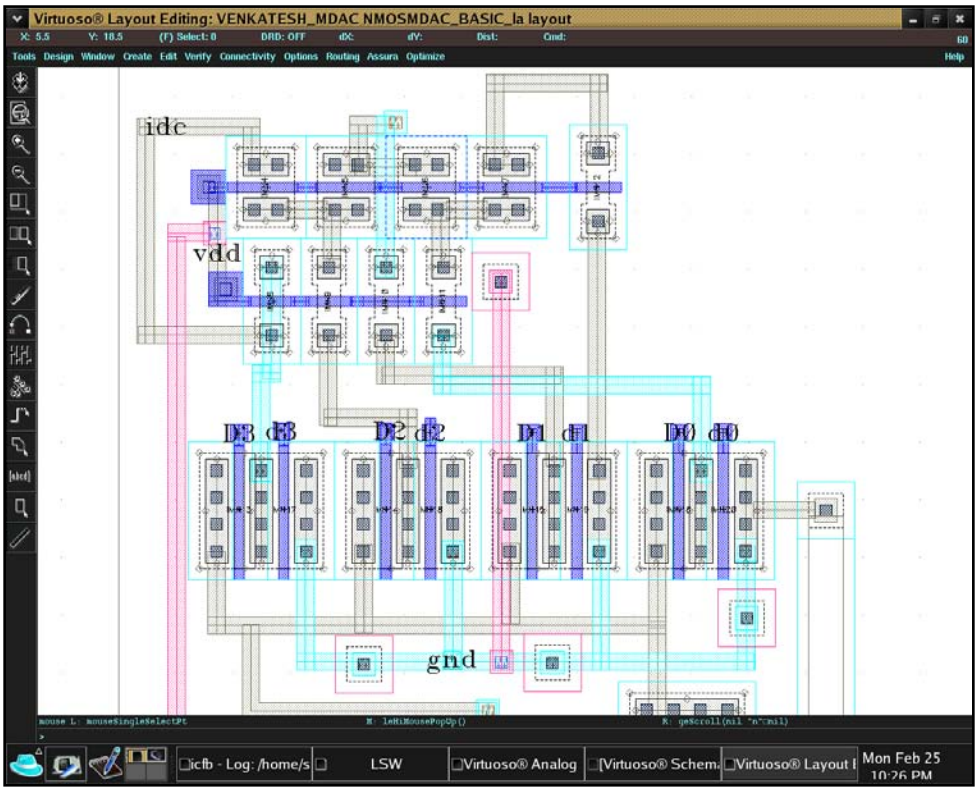


Figure 5.26 Layouts for MDAC from Virtuoso

After LVS, compare warnings and extract warnings are removed and the RC extraction is carried out. The RC extraction file is used for post layout simulation. The GDSII extracted view of the proposed MDAC is as shown in the Fig.5.27, this is carried out using the Cadence Virtuoso tool. The GDS II file extracted represents or shows that the proposed MDAC architecture is physically realizable in the form of a chip. Typically

extraction from the layout is done after the post layout simulations wherein the circuit is checked for its functionality after the interconnect resistance and capacitances are added.

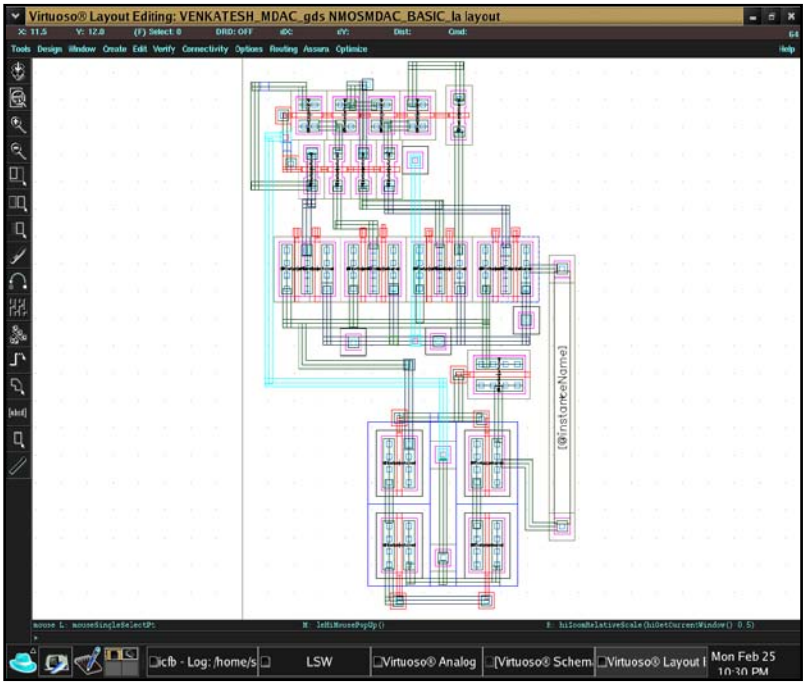


Figure 5.27 GDSII Extracted view of proposed MDAC architecture

Proposed MDAC architecture consists of 22 transistors. It is capable of computing the multiplication of analog samples with digital weights. The total area required for the hybrid network is 28 μm^2. This area excluded the storage space of the weight matrix. Every multiplier realized using hybrid architecture requires area of 28 μm^2 and 22 transistors. Total number of multipliers for the 2-D adaptive network is 128 multipliers per network (hidden layer and output layer). Total number of transistors for the multiplier stage is 2816 transistors. The area for the multiplier is 7884 μm^2. Partial product output computed need to be added in the adder circuit. As the partial products current outputs, there is no need for adder circuit. Partial product outputs are added using Kirchhoff's current law, by collecting the currents at a particular node. The current output is taken across a load to convert the compressed data into voltage samples. This is further fed into

the output layer to reconstruct the original samples. Table 5.5 compares three different multiplier cells designed in this work.

Table 5.5 Comparisons of various multiplier cells designed

Parameters	Gilbert cell	Modified Gilbert cell	New hybrid cell
Convergence time (10 MHz)	200 ns	200 ns	Not applicable
Refresh time	<20ns	<10ms	Not applicable
Power on Training	Required (5ns)	Required (8ns)	Not applicable
Circuit complexity Single cell neuron	480T, 8SW, 4C	960T, 4SW, 4C	1420T, digital memory
Storage cell	Capacitor	Capacitor	Digital memory
Accuracy	±0.3 V	±0.35 V	±0.08 V
Parameters	Gilbert cell	Modified Gilbert cell	New hybrid cell
Technology	0.18µm	0.18µm	0.18 µm
Resolution	Not applicable	Not applicable	N bit
INL	Not applicable	Not applicable	< 0.5 LSB
DNL	Not applicable	Not applicable	<0.5 LSB
Supply voltage	3.3 V	3.3 V	2.5V
Power consumption	865 mW	631 mW	232 mW
Full scale current / voltage	±1.8 V	±1.8 V	65.535 µA
Operating frequency	50 MHz	50 MHz	200 MHz

Gilbert cell multiplier presented in Fig. 5.3, modified Gilbert cell with weight updating scheme and the new hybrid cell. Gilbert cell and modified Gilbert cell multiply analog inputs with analog weights and produce analog output. The weight and bias for the network is computed during online training, and stored on a capacitor for processing the analog input samples. In the new hybrid cell multiplier, the input is analog, and the weight matrix is digital. Weights and biases obtained during training phase is digitized and stored in ROM. The performances of all the three multiplier blocks are compared. Convergence time is the time taken by the network to converge to achieve zero error during the training phase. This is the total time required to obtain the optimum weight. Refresh time is the time has be refreshed to weights, as the weights are stored on capacitor, capacitor needs to be refreshed every time to hold the optimum weight

obtained during training phase. Whenever the network is power on, it requires some time to refresh; this time is power on training. Number of transistors, number of capacitors and number of switches are used to express the complexity of the switch. Accuracy defines the minimum voltage the circuit can produce.

New hybrid cell (modified DAC) designed using NMOS transistors work at 200MHz of clock frequency, consumes power of 232mW, maximum full scale current of 65.535 µA and requires 1420 transistors (T) with digital memory storage unit. The training is done offline and the weights are stored in ROM and hence the network consumes very less power. The only limitation of this network is that every time new weights and biases have to be used for the network to achieve better performance. New hybrid architecture is realized and the results obtained have good performances as expected. The proposed neural network architecture is realized using the hybrid cell multiplier (MDAC), adder and translinear functions. Fig. 5.28 is the proposed 2D-multilayered architecture for compression. This architecture is realized using the designed hybrid neuron cell.

Fig. 5.29 shows the output layer of the 2D multilayered neural network architecture. Using the hybrid cell multiplier, adder and translinear function the 2D-multilayered network architecture is realized. The compressor network or the hidden layer requires four neurons, and the output layer requires 16 neurons. Each neuron in the hidden layer consists of 16 multipliers, one adder and one transfer function (Tansig). Each neuron in the output layer consists of 16 multipliers, one adder and one transfer function (Purelin). Since Purelin is a linear function this is not used in the output layer. Hybrid multipliers or MDAC circuit is used to realize the multiplier, as the output of hybrid cell multiplier is current, the adder block is not required both at the hidden layer and output layer. The adder block is a node that sums all the current coming out form the multiplier circuit. The current flows through a load capacitance and thus voltage is measured across the load capacitance.

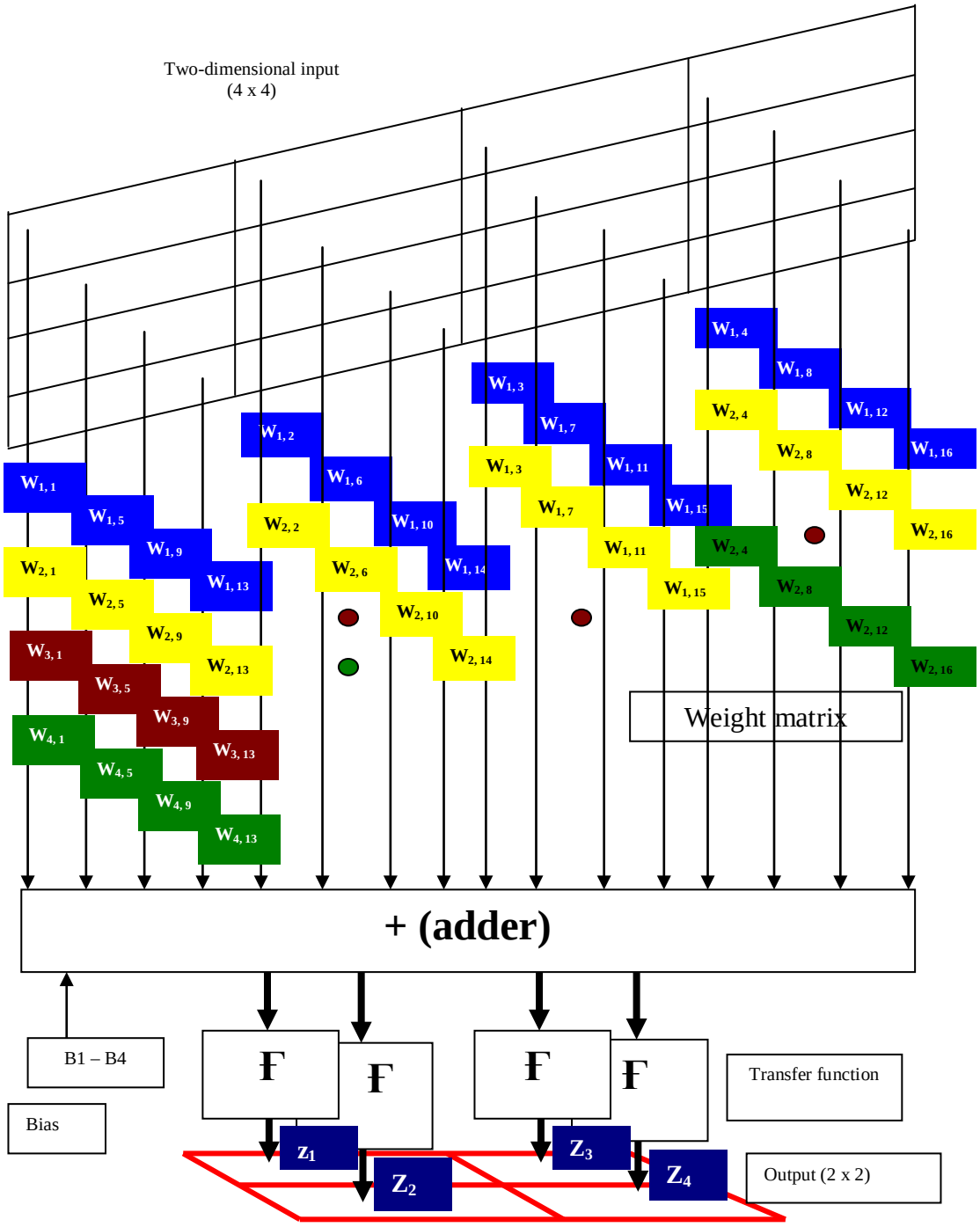


Figure 5.28 2D-multilayered architecture (hidden layer)

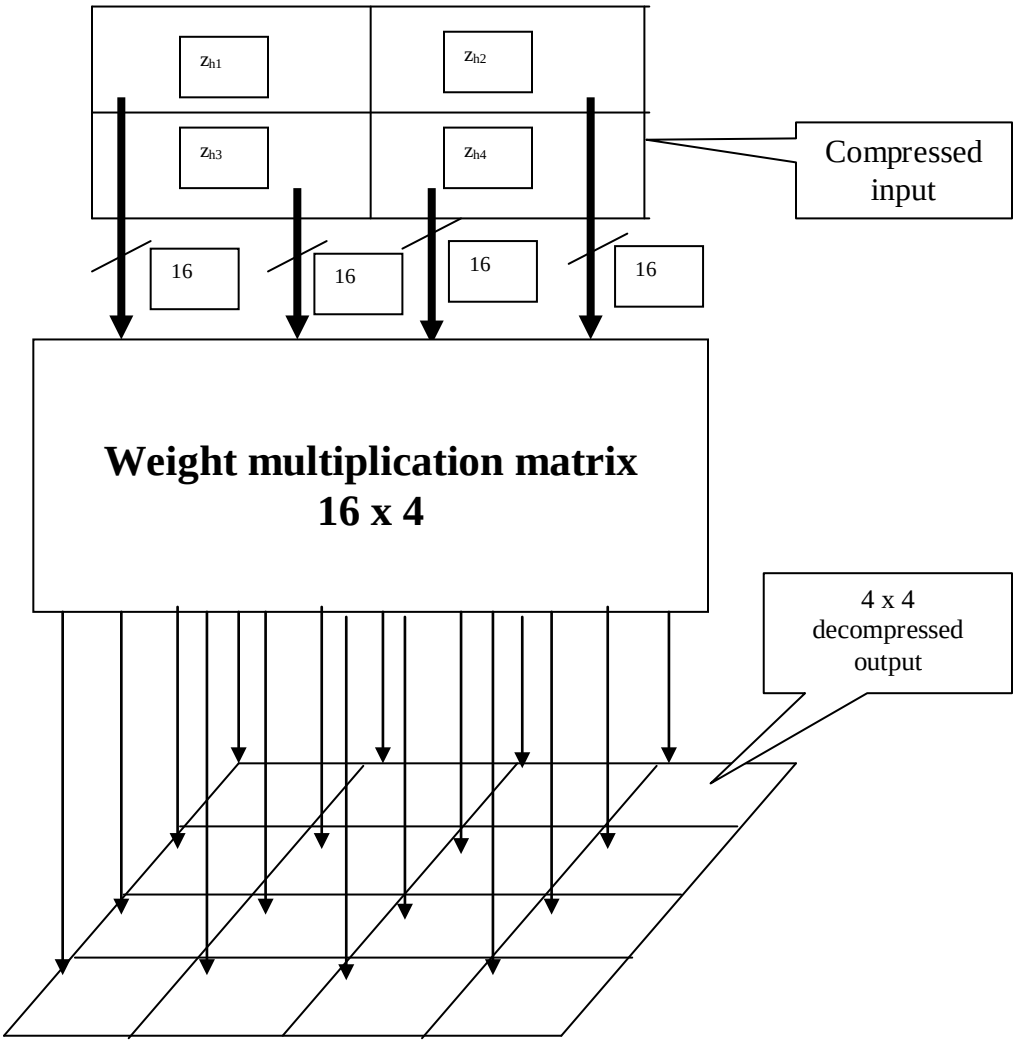


Figure 5.29 2D-multilayered architecture (output layer)

The output of the current to voltage converter at the hidden layer is given to the transfer function which is realized using the circuit (NAF) shown in Fig. 5.6 having the transfer function shown in Fig. 5.8. In the next section, design of single neuron cell is presented. Properties of this cell are analyzed based on simulation results. Multiple single neurons are integrated to model the 2D-multilayered neural network architecture.

5.9 Top-level Block Diagram of Single Neuron using Hybrid Multiplier

In this section, top level architecture of single neuron using the hybrid cell multiplier, adder and the neuron activation function is presented. The p-inputs of neuron, X_0, X_1… X_{p-1} shown in Fig. 5.30, are multiplied by the p number of synaptic weights, W_0, W_1…, W_{p-1}. The weighted sum is then forwarded to the neuron output via a nonlinear activation function S (.). Neuron output Y is then given by equation (5.1).

$$Y = S\left(\sum_{i=0}^{p-1} X_i W_i\right) \tag{5.1}$$

Multiplication operation of X_iW_i and the addition $\Sigma\ X_iW_i$ are the two arithmetic operations performed by the neuron. The implementation of the addition is easy if outputs of the synapse are currents. It is performed when synapse outputs are connected together according to Kirchhoff's Current law (KCL).

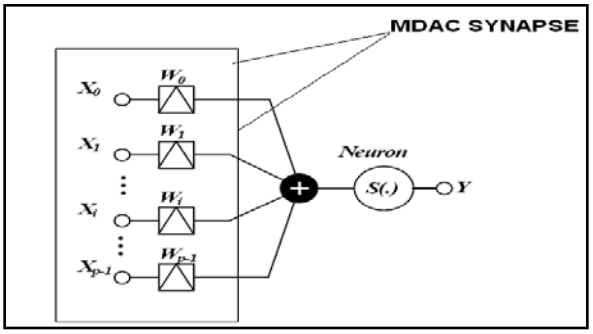


Figure 5.30 Neuron Cell

The hybrid cell (MDAC) synapse designed performs the multiplication of the analog input with the digital weights W_1 to W_{p-1} , The analog input is in the form of the current and is multiplied accordingly with the value of the digital weights represented by D_3 to D_0 and the current as I_{in}. The current I_{in} is given through the current mirror circuit, a cascoded current mirror circuit for the higher output impedance, so that it can drive large circuits for the same current values. The weighted current steering approach along with the current mirror circuit acts as the input for the DAC. The neuron circuit performs the

summation of the outputs of the different MDAC outputs, the neuron converts the currents back into voltages and this is given as the inputs for the next layer of synapses, in this proposed design of MDAC there is no need of this conversion. The digital inputs commonly known as the digital weights are stored in a memory unit from where it is loaded to the inputs of the MDAC. Fig. 5.31 shows the top level architecture of single neuron using hybrid cell multiplier, adder and neuron activation function. Hybrid multiplier is integrated with other building blocks such as adder and transfer function to form a single neuron. Weight matrix is assumed to be positive, and hence the sign bit is neglected, the hybrid multiplier has only 4-bits for weight matrix. Every hybrid cell is required to perform multiplication operation of input X_i with the weight matrix W_i. Input X_i is analog input representing pixel intensities. In order to simulate the circuit, different values of current samples are used from a current reference circuit shown in Fig. 5.22.

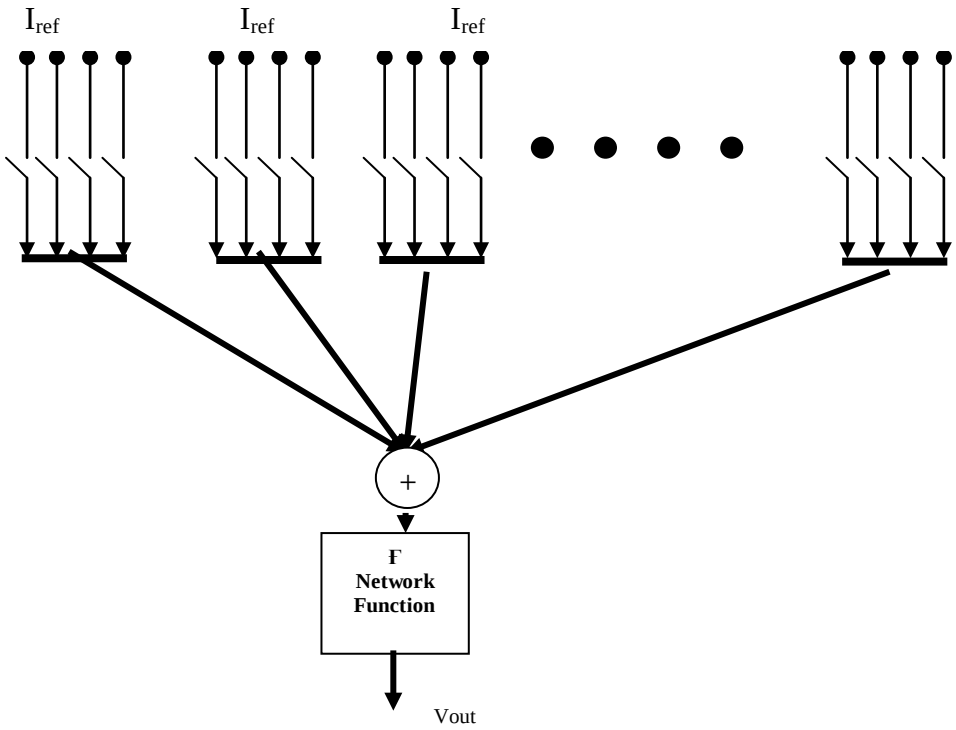


Figure 5.31 Single neuron using hybrid cell multiplier (hidden layer)

The single neuron cell shown in Fig. 5.31 is modeled using Cadence Virtuoso and simulated using Spectre targeting 180nm Technology. The binary inputs for all the multipliers have been considered from 0000 to 1111. The output of the hybrid multiplier is added and transferred using the nonlinear transfer function i.e. the NAF function. For different values of input weights varying from 0000 to 1111 and using the reference current of 16 µA, the network is simulated. The maximum current form each hybrid cell is measured is found to be 28.2 µA, the outputs of all the 16 cells are added and the maximum output current is found to be 451.2 µA. The activation function requires a voltage input in the range -3 V to +3 V, hence the current output is converted to voltage using a current to voltage converter realized using a current mirror load circuit (Appendix D describes the spice code for measuring output voltage).

5.10 Test Setup to Evaluate Performance of 2-D Network Architecture

Image pixels used to test the software reference model is equivalently represented using current equivalents. The weight matrix obtained from the software reference model is represented in digital form to test the hardware designed. Current mirrors are modeled as pixel intensities. Current mirrors are connected to the designed 2-D network architecture to compress the pixels to 2 x 2 and to further decompress to 4 x 4. The experimental setup is shown in Fig. 5.32.

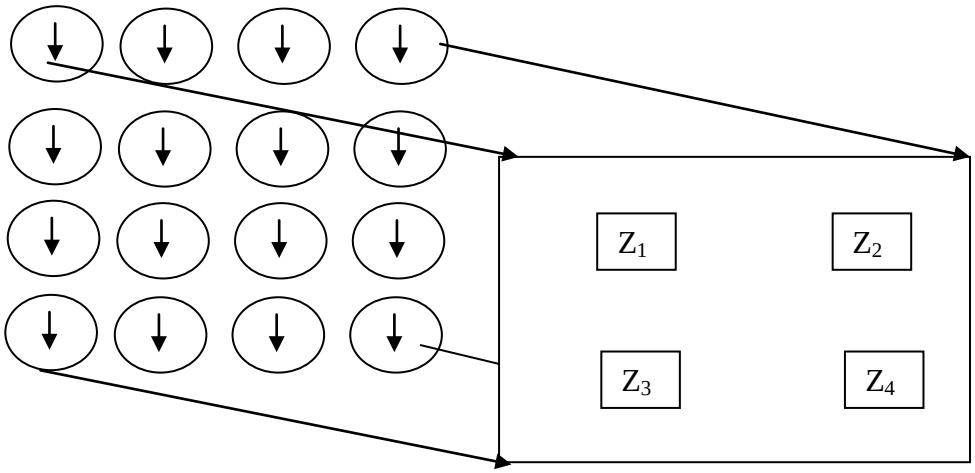


Figure 5.32 Hidden layer test setup

The pixel intensities used in the software reference model have values varying from 0 to 255; these values are normalized and are scaled from 0 to 1. To represent each of these pixel values in terms of current values, different current mirror circuits are designed that are driven from current references. The current mirror circuits provide current values varying from 0 to 1275 µA. To simulate the image sensor values, these current mirrors are arranged in 4 x 4, 8 x 8, 16 x 16, 32x 32 and 64 x 64. Same sized images were used in software reference model. Based on the test setup shown in Fig. 5.32, simulation of the proposed single neuron network shown in Fig. 5.31, the results are tabulated in Table 5.6.

Table 5.6 Simulation results of single neuron cell

1	2	3	4	5	6	7	8
Binary input	Pixel Equivalent	Hybrid multiplier output in µA	Adder output in µA	Neuron output in mV (Simulated)	Neuron output in mV (Theoretical)	Equivalent pixel values	Software reference results
0000	0	0	0	0	0	0	0
0001	008	01.7	027.2	0070.20	0081	112	116
0010	016	03.6	057.6	0149.76	0172	134	140
0011	024	05.2	083.2	0216.32	0749	142	152
0100	032	07.5	120.0	0312.00	0360	149	153
0101	040	08.7	139.2	0361.92	0417	153	161
0110	048	10.5	168.0	0436.80	0501	160	166
0111	056	12.1	193.6	0503.36	0579	168	174
1000	072	16.0	256.0	0665.60	0768	174	180
1001	080	16.7	267.2	0694.72	0802	182	191
1010	088	18.8	300.8	0782.08	0906	191	201
1011	096	20.0	320.0	0832.00	0960	198	207
1100	104	23.0	368.0	0956.80	1101	206	216
1101	112	23.9	382.4	0994.24	1140	213	221
1110	120	26.8	428.8	1114.88	1286	225	232
1111	128	28.6	451.2	1173.12	1314	236	245

The results obtained are compared with expected results and software reference model presented in chapter 4. For the circuit shown in Fig. 5.31, the input to the hybrid cells is varied from 0000 to 1111, and the outputs obtained are tabulated. In Table 5.6,

column 1 represents the binary input applied to the entire hybrid cell (16 multipliers). Column 2 is the pixel values used as test case. Column 3 is the output of the hybrid cell and column 4 is the output of the adder. The output of the adder is converted to voltage using a current mirror load designed as a load circuit having resistance of 3K Ohm (column 5). Column 6 is the expected voltage output of the single neuron cell. Column 7 is the equivalent pixel values for the simulated results and column 8 is the pixels values obtained using software reference model.

Fig. 5.33 presents the comparison of the simulated single neuron cell with theoretical values for inputs varying from 0 to 16. From the results, it is found that the single neuron cell out variation matches with the expected results. The maximum expected output when all the inputs applied to the inputs of the neuron are at maximum value of 16 is 1.31 V. Simulated results show that the maximum output voltage is 1.26 V. The error difference of 0.04 V is acceptable, and can be minimized.

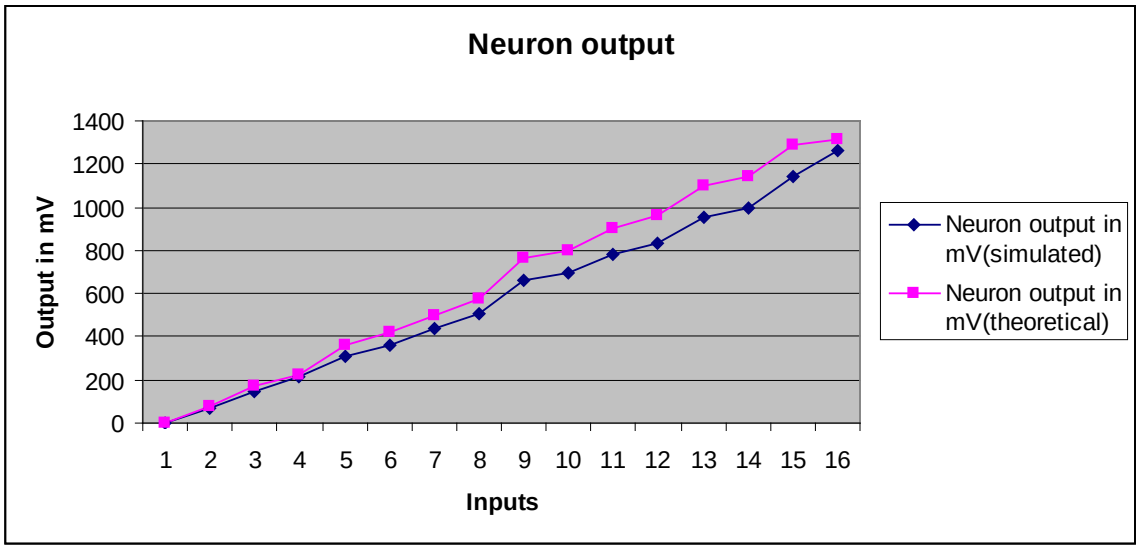


Figure 5.33 Simulated results of single neuron cell

Fig. 5.34 shows the comparison of expected pixel values and the simulated output. As the network produces voltage output, in order to compare with the software reference results, the outputs have been converted to its equivalent pixel representation and are compared with the results obtained using software reference results.

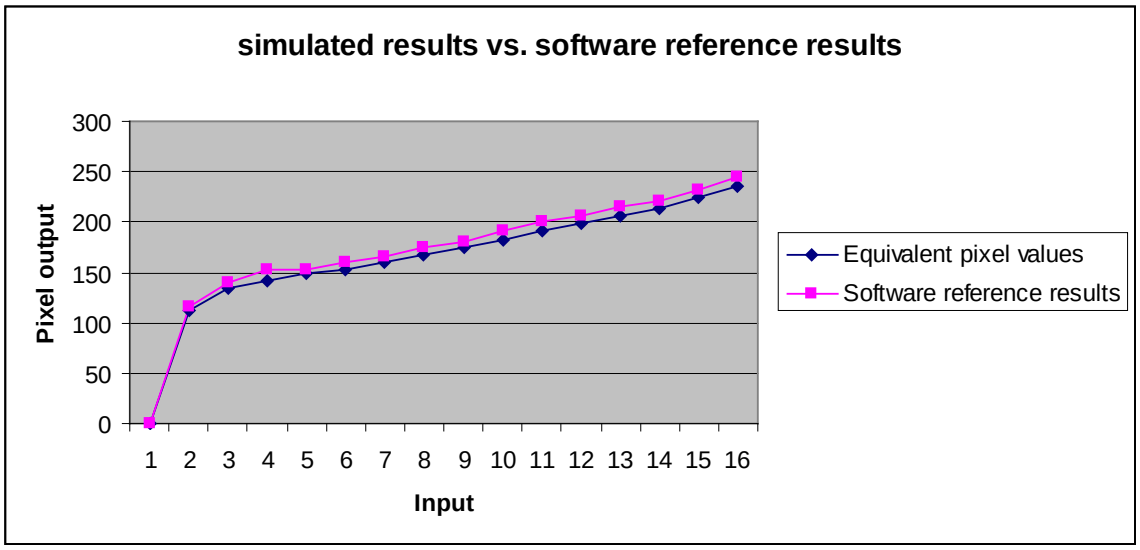


Figure 5.34 Comparison of simulated results with software reference results

Hardware simulations of the proposed design are used to test image compression and decompression of various image sizes. MSE is calculated, the results show that the hardware simulated results are very satisfactory as they achieve very less MSE. However, the complexity is that due to limitations in test setup, performance of the network is not evaluated for large size images. The decompressed network output producing current values were compared with input current values; using mathematical equation MSE and PSNR were numerical calculated. The results of the same are shown in Table 5.7.

Table 5.7 Image size and performance parameters

Image size	MSE	CR
64X64	21.2	50
32x32	18.6	50
16x16	19.2	50
8x8	14.3	50

From the results obtained, it is found that the network had less MSE for smaller size images; this is due to the fact that smaller size images have minimum number of 4 x 4 blocks and hence numbers of boundaries are less between these blocks. Large image sizes having more boundaries, affects the MSE. This can be avoided by taking

overlapping blocks of 4 x 4. This avoids boundaries and hence good MSE. The network is also tested for various compression ratios. 4 x 4 blocks of image or current mirrors were used to test the network. Different network sizes were chosen to achieve different compression ratios. The results obtained were matching the software response curve for the same compression ratios.

Table 5.8 Compression ratio and MSE

Network size	Size of hidden layer	BPP	MSE
16-8-16	8	4	12.43
16-6-16	6	3	18.4
16-5-16	5	2.5	19.8
16-4-16	4	2	14.6
16-2-16	2	1	15.7
16-1-16	1	0.5	17

Table 5.8 shows the results of compression ratio vs. MSE for the software simulated hardware reference model. At higher compression, MSE is 17, and at lower compression it is 12.43.

In this chapter, the hardware model for the proposed network is designed, modelled, simulated using Cadence Virtuoso and Synopsys HSpice EDA tools. The basic building blocks for the neuron model is first modelled using spice coding, spice models are verified using 180nm and 130nm library. The spice results are set as reference models. Based on the designed schematic, schematic capture of the proposed neuron is carried out using Virtuoso schematic editor. Simulation results are verified with spice models. Finally, layouts for the neuron cell are designed using Virtuoso Layout Editor, layouts are verified for LVS, DRC violations. Parasitic extraction of the layout is performed using RC-XT and spice simulations of the extracted model are carried out. Results obtained are verified with spice model results. As industry standard tools that are termed as signoff tools used to verify the design, the results presented in this chapter are almost hardware worthy. The next chapter summarizes the research work and highlights the major conclusions and scope for future work.

Chapter 6 – Conclusion

6.1 Conclusion and Recommendations for Future Work

In this research work, adaptive two-dimensional multilayer neural network architecture has been proposed, designed, modelled, simulated and verified using both the software and hardware models. The adaptive architecture proposed automatically selects one of the three 2-D multilayered neural network architectures based on image entropy and Bits Per Pixel (bpp). This architecture eliminates the need for 2-D to 1-D reordering of image samples and also eliminates the need for analog to digital conversion of image intensities. A modified backpropagation algorithm that is suitable to train the 2-D multilayer network architecture is proposed and is used to train the TDMNN architecture. Software reference model for Adaptive TDMNN architecture is developed. MSE, PSNR and Maximum Error for various images are computed using the developed software reference model. Network parameters such as the number of hidden layers, number of neurons in each layer, input sub-block image size, network functions and compression ratio are estimated based on results obtained. Based on the experimental results it is concluded that a two layered network with sub-block of image size 4 x 4 is optimum in terms of network performance and computation time. Input block size restricted to 4 x 4 was selected as a tradeoff between complexity and quality. Noise analysis carried out on TDMNN shows that the network has 2 to 25 times improvement compared to DWT-SPIHT technique. Error analysis of the TDMNN architecture reveals 10 to 30 times improvement over DWT-SPIHT technique.

Initially TDMNN architecture was designed for image compression and decompression. Performances of three TDMNN architectures (Linear, Nonlinear and Hybrid) were analyzed. From the results it was found that for 0.5 bpp hybrid networks achieves better PSNR and MSE compared to linear network. At 7.5 bpp, linear network performs better than hybrid network. For bpp between 2 and 5, the hybrid network achieves better performance compared to linear network. Also the nonlinear network was able to achieve better results compared to the linear and hybrid networks. Based on

the results, it was concluded that the network performance is a function of the image. Hence, in order to achieve better performance compared to the conventional techniques, adaptive TDMNN architecture is proposed. In this architecture, Entropy of input image is computed. Based on the Entropy and the required compression ratio, the control unit automatically selects appropriate TDMNN architecture for image compression and decompression. Adaptive TDMNN architecture is three to ten times better than TDMNN in terms of quality metrics such as MSE and PSNR. MSE and PSNR results obtained for adaptive TDMNN architecture for 4 bpp and less than 4 bpp reveals three times better compared with DWT-SPIHT results. Software simulation results show that Adaptive TDMNN architecture is 60 times faster than DWT-SPIHT. Network is trained with multiple image data sets to generalize the network for compression of various images.

Based on the software reference model developed, basic building blocks for Adaptive TDMNN architecture are identified for VLSI implementation. Multipliers, adders and network functions are the three major building blocks for adaptive TDMNN architecture. VLSI implementation of TDMNN building blocks are carried out using industry standard tools. Analog network architecture is designed and implemented for image compression and decompression. Three different neuron cells (Gilbert cell based neuron, modified Gilbert cell based neuron and hybrid neuron cell) have been designed and analyzed for its area, timing and power performances. Hybrid neuron cell is selected as it is found to be more suitable for VLSI implementation. Hybrid cell multiplies analog samples with digital weights and hence called as hybrid neuron cell. Hybrid neuron cell consists of multiplier, adder and network function. There is a need of 128 multipliers per network for adaptive TDMNN architecture (hidden layer and output layer). A modified Multiplying DAC architecture based multiplier is designed involving a total number of 2816 transistors. The area for the multiplier is 7884 μm^2. Hybrid cell (based on modified MDAC) is designed using NMOS transistors work at 200 MHz of clock frequency. It consumes 232 mW of power at the maximum full scale current of 65.535 µA. Weights

and biases obtained during training are stored in Read only Memory (ROM). Test setup for verifying the TDMNN architecture as compressor and decompressor is designed. Image sizes 64 x 64 to 8 x 8 were used to test the network performance. MSE computed for different image sizes varies from 21 to 14. These results were validated against the software reference model and the difference between hardware and software models was less than 10%. The network was tested for various compression ratios and the results obtained were found to agree with software reference model. A full chip design of the proposed architecture was implemented using Cadence Virtuoso, and the physical verification was carried out using Assura. DRC and LVS checks were performed and then GDSII was generated for chip fabrication.

Recommendations for Future Work:

1. Network performances of Adaptive TDMNN can be further improved by identifying the entropy of sub-images and selecting the required network. Based on image entropy, bpp and computation time, sub-image size can be selected for compression using Adaptive TDMNN.
2. The network performances can be further improved by having multiple two-dimensional network architectures supporting multiple image properties. Also, robust classification algorithms can be identified to select the required network for specified compression and decompression.
3. Two-dimensional training algorithms can be identified to train two-dimensional network, as this research while extends the 1-D training algorithm to 2-D network by rearrangement of the input image.
4. The proposed network can be further extended to perform image segmentation, edge detection and restoration replacing the digital weight matrix and hence can achieve reconfigurabilty.
5. Hardware blocks designed can be further optimized my minimizing the weight matrix by exploiting the redundancy in the weight matrix.

6. Hardware realization on silicon and real time testing of the proposed model would be a very good scope for future work.
7. The compressed analog data being analog can be further converted to digital and two dimensional encoding schemes and can be used compress the images and higher compression can be achieved. The recommendations lead to two-dimensional signal processing techniques that are slowly finding prominence for next generation communication technologies.

List of Publications

National Conferences:

- Cyril Prasanna Raj, P. and Pinjare, S.L. (ed.) (2005) *NN for image compression On FPGA*. 'National Conference on IMS'. held 26 March 2005 at Sona Institute of Technology. Salem (Won the best paper award)
- Cyril Prasanna Raj, P. and Pinjare, S.L. (ed.) (2005) *Multilayered Neural Network on FPGA*. 'IEEE National Conference on VLSI, Embedded, DSP applications (VEDAS)'. held 3-4 June 2005. Salem (Won the best Paper award)

International Conferences:

- Cyril Prasanna Raj, P. and Pinjare, S.L. (ed.) (2005) *Pipelined and Parallel Architectures for MLNN on FPGA*. 'International Conference on Advances in Computer Vision and Information Technology'. held 16-17 October 2005 at NIT(Trichy). Cochin
- Cyril Prasanna Raj, P. and Ramanaiah, K.V. (ed.) (2008) *FPGA Implementation of DWT and Neural Network Hybrid Architecture for Image Compression*. 'IEEE conference on Biomedical Signal Processing'. held 3-6 December 2008. Singapore
- Cyril Prasanna Raj, P. and Pinjare, S.L. (ed.) (2009) *ASIC Implementation of Nonlinear Neural Networks for Image Compression Optimizing Area and Power*. 'International Conference on Intelligent Systems and Control'. held 6-7 February 2009 at Karpagam University. Coimbatore
- Cyril Prasanna Raj, P. and Pinjare, S.L. (ed.) (2009) *Analog VLSI Implementation of Novel Hybrid Neural Network Multiplier Architecture.* 'International Conference on Intelligent Systems and Control'. held 6-7 February 2009 at Karpagam University. Coimbatore
- Cyril Prasanna Raj, P. and Pinjare, S.L. (ed.) (2009) *Analog VLSI Implementation of 2-D Multidimensional Architecture for Image Compression.* 'International Conference on Intelligent Systems and Control'. held 6-7 February 2009 at Karpagam University. Coimbatore

Journal Publication:

- Cyril Prasanna Raj, P. and Pinjare, S.L. (2009) 'Design and Analog VLSI Implementation of Neural Network Architecture for Signal Processing'. *European journal of scientific research* 27 (2)

Papers Submitted:

- Cyril Prasanna Raj, P. and Pinjare, S.L. Vershinin, Y.A. (2009) 'Analog Neural Architecture with Gilbert Cell Supporting Large Voltage Swing'. *I-Managers Journal for Nano sciences*, India
- Cyril Prasanna Raj, P. and Pinjare, S.L. Vershinin, Y.A. (2009) 'Analog VLSI Implementation Of Neural Networks For Signal Processing And Image Compression'. *Australian Journal for Electrical and Electronics*
- Cyril Prasanna Raj, P. and Pinjare, S.L. Vershinin, Y.A. (2010) '2-D Multilayered Analog Neural Network Architecture for Image Compression'. *European Journal of Scientific Research*
- Cyril Prasanna Raj, P. and Pinjare, S.L. Vershinin, Y.A. (2010) '2-D Multilayered Neural Network Architecture for Image Compression'. *WSEAS Transactions on Signal Processing*

Patents Submitted for Review:

- Multiplierless hybrid neural network architecture for image compression and decompression
- Two-dimensional Training for 2-D multilayered neural network architecture for image compression and decompression
- Two-dimensional multilayered neural network architecture for image compression and decompression for telemedicine applications
- Two-dimensional reconfigurable neural network based image processing unit using Multicore architectures
- 3D-DWT and TDMNN architecture for Image Processing
- Adaptive TDMNN architecture for video processing

REFERENCES

Aaron Ferrucci (1994) *A field-programmable gate array implementation of a self adapting and scalable connectionist network.* Unpublished Master's thesis. Santa Cruz: University of California

Adnan and Dimililer (2008) 'Image compression using neural networks and harr wavelet'. *WSEAS Transactions on Signal Processing* 4, 330 – 339

Ali Naderi, Adbollah Koehi, Khayrollah Haidi and Hadi Ghasemzadeh (2008) 'A new high speed and low power four-quadrant CMOS analog multiplier in current mode'. Int. Journal of Electronic Communication 63, 769-775

Amerijckx, C., Legaty, J. D. and Verleysenz, M. (2003) 'Image Compression Using Self-Organizing Maps'. *Systems Analysis Modeling Simulation* 43 (11), 4230-4235

Andra, K., Chakrabati, C. and Acharya, T. (2003) 'A High-Performance JPEG2000 Architecture'. *IEEE Trans. on Circuit and Systems for Video Technology*13 (3) 209-218

Andreas, G. and Boahen, A. (1996) 'Translinear Circuits in Sub threshold MOS'. *Analog Integrated Circuits and Signal Processing* 9,141-166

Anne Vanden Bosh , Marc Borremans, A. F., Michel, S. J and Willy Sasen (2001) 'A 10-bit 1-Gsample/s Nyquist Current-Steering CMOS D/A Converter'. *IEEE Journal of Solid State of circuits* 36 (3), 315-323

Anthony, D., Taylor, E., D. and Barham, J. (1989) 'A study of data compression using neural networks and principal component analysis'. *Colloquium on Biomedical Applications of Digital Signal Processing*, 1–5

Arthur, E. B. and Yu-Chi, H., (1968) Applied optimal control: optimization, estimation, and control. Blaisdell Publishing Company or Xerox College Publishing, 481

Auda, G. and Kamel, M. (1999) 'Modular neural networks: a survey'. *International Journal of Neural Systems* 9 (2), 129-151

Backus, J. (1978) 'Can programming be liberated from the von Neumann style'. *Communications of the ACM* 21 (8), 613-641

Barni, M., Bartolini, F. and Piva, A. (2001) 'Improved Wavelet-Based Watermarking Through Pixel-Wise Masking'. *IEEE Trans. on Image Processing* 10 (5), 783-791

Becker, S. and Plumbley, M. (1996) 'Unsupervised neural network learning procedures for feature extraction and classification'. *International Journal of Applied Intelligence* 6 (3), 185-203

Bernabe, L. (1992) 'A Modular T-Mode Design Approach for Analog Neural Network Hardware Implementations'. *IEEE Journal of Solid-state Circuits* 27 (5), 701-713

Beuchat, J.-L., Haenni, J.-O. and Sanchez, E. 'Hardware Reconfigurable Neural Networks'. *Parallel and Distributed Processing* IPPS/SPDP'98, Lecture Notes in Computer Science, Springer-Verlag, Vol. 1388, pp. 91-98, 1998.

Blake, J. and McDaid, L. (2005) 'Using Xilinx FPGAs to Implement Neural Networks and Fuzzy Systems'. Faculty of Engg, Univ. of Ulster, Magel College, Northland Rd. Derry

Blelloch, G. E. (1990) *Vector Models for Data-Parallel Computing*. Cambridge: MIT Press

Benbenesiti, Y. (1997) 'New simple three-layer neural network for image compression'. *Optical Engineering* 36, 1814-1817

Benbenesiti, Y., Kornreich, D., Mitchell, H.B., Schaefer, P.A. (1999) 'Fixed bit-rate image compression using a parallel-structure multilayer neural network'. *IEEE Trans. on Neural Networks* 10, 1166 – 1172

Bogdan, W., Serdar, I., Okyay, K and Onder, M. (1997) 'An Algorithm for Fast Convergence in Training Neural Networks'. 3, pp.345-351

Bose, N. K. and Liang, P. (2002) *Neural Network Fundamentals with graphs, algorithms and application*. New Delhi: Tata McGraw hill

Bugeja, R. and Song, B. S. (2000) 'A self-trimming 14-b 100-MS/s CMOS DAC'. *IEEE Journal of Solid-State Circuit* 35, 1841-1852

Burcu Kapanogulu and Tulay Yildirim (2003) *A novel four quadrant analog multiplier for artificial neural networks.* 'International Turkish Symposium on Artificial Intelligence and Neural Networks'. held 2003 at Turkey

Carrato, S. (1992) 'Neural networks for image compression'. In *Neural Networks: Advances and Applications*. Gelenbe Publications, 177-198

Carrato, S. Ramponi, G. (1991) *Improved structures based on neural networks for image compression*. 'IEEE Workshop on Neural Networks for Signal Processing'. held Sept.29-Oct.2, 1991 at New Jersey, USA, 493-502

Chang, IS (2007) Space launch vehicle reliability [online] available from <www.aero.org/publications/crosslink/winter2001/03.html> [Feb 2009]

Chi-Hung Lin and Klaas Bult (1998) 'A 10-b, 500-MSample/s CMOS DAC in 0.6 mm^2'. *IEEE journal of solid-state circuits* 33, 1948-1958

Chiang, J. S., Lin, Y. S and Hsieh, C. Y. (2002) 'Efficient Pass-Parallel Architecture For EBCOT in JPEG2000'. *IEEE International Symposium on Circuits and Systems* 1, 773-776

Chrysafis, C. and Ortega, A. (1998) 'Line based, reduced memory, wavelet image compression'. *Proceedings of Data Compression Conference*, 398–407

Chun Lu, Bing-xue Shi and Lu Chen (2002) Hardware Implementation of an Analog Accumulator for On-chip BP Learning Neural Networks, Institute of Microelectronics, Tsinghua University Beijing, China

Conforto, S. (1995) 'High-quality compression of echo graphic images by neural networks and vector quantization'. *Medical and Biological Eng. Computation* 33, 695-698

Cottrell, G. W. and Metcalfe, J. (1996) 'EMPATH: Face, emotion, and gender recognition using holons'. *Advances in Neural Information Processing Systems* 3, 564–571

Costa, S. and Fiori, S. (2001) 'Image compression using principal component neural Networks'. *Image and Vision Computing* 19, 649-668

Cottrell, G. W. and Munro, P. (1988) 'Principal components analysis of images via back Propagation'. *SPIE, Visual Comm. and Image Processing* 1001, 1070–1077

Cox, C. E. and Blanz, Ganglion, W. E. (1992) 'A fast field-programmable gate array implementation of a connectionist classifier'. *IEEE Journal of Solid State Circuits* 27, 288-299

David, T. and Michael, M. (2001) 'Image Compression Fundamentals: Standards and Practice'. *The Kluwer International Series in Engineering and Computer Science*

Dony, R. D. and Haykin, S. (1995) 'Neural Network Approaches to Image Compression'. *In Proceedings of IEEE* 83 (2), 288-303

Eldridge, J. G. and Hutchings, B. L. (1994) *FPGA density enhancement of a neural network through run-time reconfiguration*. Unpublished Master's thesis. Brigham: Young University

Eldridge, J. G. and Hutchings, B. L. (1994a) Density enhancement of a neural network using FPGAs and run-time reconfiguration, IEEE Workshop on FPGAs for Custom Computing Machines, (Apr. 10-13), pp.180-188

Eldridge, J. G. and Hutchings, B. L. (1994b) Rann: A hardware implementation of the backpropagation algorithm using reconfigurable FPGAs. '*IEEE International Conference on Neural Networks'*. held Jun 26-Jul 2. Orlando: FL

Encyclopedia (2001) Error analysis in lossy compressed image transmission algorithms [online] available from <http://www.encyclopedia.com/doc/1G1-146892300.html> [24 July 2005]

Egmont, P., Ridder and Handels (2002) 'Image Processing with Neural Networks – a review'. *Pattern Recognition* 35, 2279- 2301

Eric Vittoz, A. (2003) 'Weak Inversion in Analog and Digital Circuits'. *CCCD Workshop*, Lund

Fang, H. C., Wang, T. C., Lian, C. J., Chang, T. H. and Chen, L. G. (2003) 'High Speed Memory Efficient EBCOT Architecture for JPEG2000'. *IEEE International Symposium on Circuits and Systems* 2, 736-739

Fethi, B., Gokcen, I. and Qidwai, U. (2001) Chaotic gray-level image transformation, Journal of Electronic Imaging, 14 (04)

Fisher, R., Perkins, S., Walker, A. and Wolart, E. (2003) Pixel connectivity [online] available from <http://homepages.inf.ed.ac.uk/rbf/HIPR2/> [May 2011]

Frescura, F., Giorni, M., Feci, C. and Cacopardi, S (2003) 'JPEG2000 and MPEG2000 transmission in 802.11 wireless local area networks'. *IEEE Trans. on Consumer Electronics* 9 (4), 861–871

Garris, M. D., Wilson, C. L. and Blue, J. L. (1998) 'Neural network based systems for Hand print OCR applications'. *IEEE Trans. On Image Processing* 7, 1097–1112

Golomb, B. A., Lawrence, D. T. and Sejnowski, T. J. (1996) 'SEXNET: A neural network identifies sex from human faces'. *Advances in Neural Information Processing Systems* 3, 572– 577

Greenhil, D., Davies, E. R. (1994) 'Relative effectiveness of neural networks for image noise suppression'. *In Proceedings of the Pattern Recognition* 4, Vlieland, 367–378

Guccione, S. A. and Gonzalez, M. J. (1993) 'A data parallel programming model for reconfigurable architectures'. *IEEE workshop on FPGAs for custom computing machines*, 78-79

Guan, L., Anderson, J. A. and Sutton, J. P. (1997) 'A network of networks processing model for image regularization'. *IEEE Trans. Neural Networks* 8, 169–174

Hadi and Mansour (2009) 'A complexity based approach in image compression using neural networks'. *International Journal on Signal Processing* 5 (2), 82–92

Hanek, H. and Ansari, N. (1996) 'Speeding up the generalized adaptive neural filters'. *IEEE Trans. on Image Processing* 5, 705–712

Hahn-Ming Lee, Chih-Ming Cheb and Tzong-Ching Huang (2001) Learning improvement of back propagation algorithm by error saturation prevention method, International Joint Conference on Neural Networks 3, 1737 – 1742 Washington DC

Heinrich, N. and Wu, J. K. (1993) 'Neural network adaptive image coding'. *IEEE Trans. On Neural Networks* 4 (4), 605-627

Hillis, W. D. (1985) *The Connection Machine*. Cambridge: MIT Press

Hussein, C. (1997) 'Analysis and Design Of Analog Microelectronic Neural Network Architectures With On-Chip Supervised Learning'. Unpublished PhD Thesis. University of Genoa

Huynh-Thu, Q., Ghanbari, M. (2008) 'Scope of validity of PSNR in image/video quality Assessment'. *Electronics Letters* 44, 800–801

Hyun-Ho Cho, Cheong-Yong Park, Gun-Shik Yune and Kwang-Sub Yoon (2004) 'A 10-Bit 210MHz CMOS D/A Converter for WLAN'. *IEEE Asia-Pacific Conference on Advanced System Integrated Circuits* 5, 106-109

Iain, E. G. (2002) 'Video Codec Design, Developing Image and Video Compression Systems'. John Wiley & Sons

Ikegami, T. (2008) Amphion IP to decode MPEG2 HDTV (4:2:2) on FPGA [online] available from <http://www.amphion.com/cs6510.html> [Jan 2009]

Ivan, V. (2006) 'An experience in image compression using neural networks'. *48th international symposium, ELMAR*, 95-98

Jacob Wikner and Nianxiong Tan (1999) 'Modeling of CMOS Digital-to-Analog Converters for Telecommunication'. *IEEE Trans. on Circuits and Systems, Analog and Digital Signal Processing* 46 (5), 489-499

Jiang, J. (1999) 'Image Compression with Neural Networks - A Survey'. *IEEE Trans. On Signal Processing and Image Communication* 14, 737- 760

Jianxun and Huang (2004) 'Image Compression Using Principal Component Neural Network'. *8th International Conference on Control, Automation, Robotics and Vision* 1, 698-701

Jiang, J. (1995) 'Algorithm design of an image compression neural network'. *Proc. Of World Congress on Neural Networks*, 1792-1798

Jiang, J. (1995a) 'A novel design of arithmetic coding for data compression'. *IEEE Proc. On Computer and Digital Techniques*, 419-424

Jiang, J. (1996) 'A neural network based lossless image compression'. *In Proc. Visual' 96: International Conference on Visual Information Systems*, 192-201

Jiang, J. (1996a) 'Design of neural networks for lossless data compression'. *Optical Engineering* 35, 1837-1843

Jiang, J. (1996b) 'Fast competitive learning algorithm for image compression neural Networks'. *Electronic Letters* 32, 1380-1381

Jiang, W.W., Kiang, S.Z., Hakim, N.Z. and Meadows, H.E. (1993) *Lossless compression for medical imaging systems using linear/nonlinear prediction and arithmetic coding, Proceedings of IEEE International Symposium on Circuits and Systems* 1, 283-286

JPEG2000 Final Committee Draft (FCD), *JPEG2000 Committee Drafts*. [Online]

Joe Burns (2003) *Image Formats* [online] available from <http://www.htmlgoodies.com/tutorials/web_graphics/article.php/3479931> [3 June 2004]

Kai Chen, Yuan Xue, Samarth H. Shah and Klara Nahrstedt (2001) 'Understanding Bandwidth-Delay Product in Mobile Ad Hoc Networks'. Elsevier Publications

Karayiannis, N. B. and Pai, P. I. (1995) 'Fuzzy vector quantization algorithms and their application in image compression'. *IEEE Trans. Image Processing* 4, 1193-1201

Karayiannis, N. B. and Pai, P. I. (1996) 'Fuzzy algorithms for learning vector Quantization'. *IEEE Trans. On Neural Networks* 7, 1196-1211

Karim Nice, Tracy V. Wilson and Gerald Gurevich (2003) How digital cameras work [online] available from <www.howstuffworks.com/digital-camera.htm> [3 June 2005]

Kohno, R., Arai, M. and Imai, H. (1990) 'Image compression using a neural network with learning capability of variable function of a neural unit'. *SPIE Visual Communications and Image Processing* 1360, 69–75

Koh, J., Suk, M. S. Bhandarkar, S. M. (1995) 'A multilayer self organizing feature map for range image segmentation'. *IEEE Trans. on Neural Networks* 8, 67–86

Kohonen, T. (1984) Self-Organization and Associative Memory, Springer

Kornreich, D., Benbenisti, Y., Mitchell, H.B., Schaefer, P. (1997) 'Normalization schemes in a neural network image compression algorithm', *SPIC* 10, (4), 269-278

Kornreich, D., Benbenesiti,Y., Mitchell, H.B., Schaefer, P. (1997) 'A high performance single-structure image compression neural network'. *IEEE Trans. On Aerospace Electronic Systems* 33, 1060-1063

Kotropoulos, C., Magnisalis and X. Pitas, I. (1994) ,Nonlinear ultrasonic image processing based on signal-adaptive filters and self-organizing neural networks'. *IEEE Trans. On Image Processing* 3, 65–77

Kristian, R. N. (2003) 'A Reconfigurable Computing Architecture For Implementing Artificial Neural Networks on FPGA'. Unpublished M.Sc. Thesis, The University of Guelph

Kuroki, N. (1992) 'Lossless image compression by two dimensional linear predictions with variable coefficients'. *IEICE Trans. On Fund.*, 882-889

Kwan, H. K.(1992) 'Simple sigmoid. like activation function suitable for digital hardware implementation'. *Electronic Letters* 28, 1379 – 1380

Lee, C. C. and Degyvez, J. P. (1996) 'Color image processing in a cellular neural network environment', *IEEE Trans. Neural Networks* 7, 1086–1098

Le, D. X., Thoma, G. R. and Wechsler, H. (1995) 'Classification of binary document images into textual or nontextual data blocks using neural network models'. *Machine Vision Applications* 8, 289–304

Lewicki, W. C. and Olshausen, B. A. (1999) 'Probabilistic framework for the adaptation and comparison of images codes'. *Journal of Optical Society of America A.* 16 (7), 1587-1601

Lian, C. J., Chen, K. F., Chen, H. H., and Chen, L. G. (2003) 'Analysis and Architecture Design of Block-Coding Engine for EBCOT in JPEG2000'. *IEEE Trans. on Circuit and Systems for Video Technology* 13 (3), 219-230

Lin, J. S. and Liu, S. H. (1999) 'A competitive continuous Hopfield neural network for vector quantization in image compression'. Engineering Applications of Artificial Intelligence 12 (2), 111-118

Lin, S. and Yu, P. S. (1982) 'A Hybrid ARQ Scheme with Parity Retransmission for Error Control of Satellite Channels'. IEEE Trans. on Communications 30 (7), 1701-1719

Lin, W.C., Tsao, E. C. and Chen, C.T. (1992) 'Constraint satisfaction neural networks for image segmentation'. *IEEE Trans. on Pattern Recognition* 25, 679–693

Lippmann, R. P. (1987) 'An introduction to computing with neural nets'. *IEEE ASSP Magazine* 4, 4-21

Manjunath, B. S., Simchony, T. and Chellappa, R. (1990) 'Stochastic and deterministic networks for texture segmentation'. *IEEE Trans. On Acoustics, Speech Signal Processing* 38, 1039–1049

Marshall, J. A. (1990) 'Self-organizing neural networks for perception of visual motion'. *IEEE Trans. On Neural Networks* 3, 45–74

Martin Sauter (2011) '*From GSM to LTE: AN introduction to Mobile Networks and Mobile Broadband'* Wiley Publications

Matsumoto, T., Kobayashi, and Togawa, Y. (1992) 'Spatial versus temporal stability issues in image processing neuro chips'. *IEEE Trans. On Neural Networks* 3, 540–569

Marcelo, H. (1994) 'A reconfigurable hardware accelerator for back-propagation connectionist classifiers'. Unpublished M.Sc. Thesis, Santa Cruz: University of California

Mitchell, J. L. and Pennebaker, W. B. (1993) 'JPEG Still Image Data Compression Standard'. New York: Van Nostrand

Mitra, Majid and William (2005) 'Design and implementation of multilayer mixed signal on chip neural network'. *IEEE trans. on Neural Networks* 5, 413-416

Mitra, S. and Yang, S. Y. (1999) 'High fidelity adaptive vector quantization at very low bit rates for progressive transmission of radiographic images'. *Journal of Electronic Imaging* 8, 23–35

Mougeot, M., Azencott, R. and Angeniol, B. (1991) 'Image compression with back propagation: improvement of the visual restoration using different cost functions'. *IEEE Trans. On Neural Networks* 4, 467-476

Mohammed, A. and Salameh, A. (2005) 'Speeding up Back-propagation Neural Networks'. *In Proceedings of the Informing Science and IT Education Joint Conference* Flagstaff, Arizona, 16-19

Nachtergaele, Lafruit, L., Bormans, J., Engels, M. and Bolsens, I. (1999) ,Optimal memory organization for scalable texture codecs in MPEG-4'. *IEEE Trans. on Circuits Systems for Video Technology* 9, 218–243

Namphol, A. Chin, S. and Arozullah, M. (1996) 'Image Compression with Hierarchical Neural Network'. *IEEE Trans. Aerospace and Electronic Systems* 32 (1), 326-338

Navin Saxena and James J. Clark (1994) 'A four quadrant CMOS analog multiplier for analog neural networks'. IEEE Journal of Solid Sate Circuits 29 (6), 746-749

Ngan, S. C. and Hu, X. (1999) 'Analysis of functional magnetic resonance imaging data using self-organizing mapping with spatial connectivity'. *Magazine on Resonance*

Medicals 41, 939–946

Nossek, J. A. and Roska, T. (1993) 'Special issue on Cellular Neural Networks'. IEEE Trans. On Circuits and Systems

Qiu, G., Varley, M. and Terrel, T. (1993) Image compression by edge pattern learning using multilayer perception, *Electronic letters* 29 (7), 601–603

Oja, E. (1988) 'A simplified neuron model as a principal component analyzer'. *Journal of Mathematics. Biology* 15 (3), 1432-1436

Lampinen, J., Laaksonen, J. and Oja, E. (1997) 'Neural Network Systems, Techniques and Applications in Pattern Recognition', Report B1, Laboratory of Computational Engineering, Helsinki University of Technology, 1997. (PostScript) Also published as Pattern Recognition, in C. T. Leondes (Ed.), Image Processing and Pattern Recognition, Vol. 5. in series Neural Network Systems Techniques and Applications, Academic Press, pp. 1-59, 1998.

Omer, M., Farhat, A., Momoh and Salami (2007) 'Learning algorithm effects on multilayer feed forward artificial neural network performance in image coding'. *Journal of Engg. Science and Technology* 2 (2), 88-199

Omondi, R. and Rajapakse, C. (2006) 'FPGA Implementation of Neural Networks'. U.S. :Springer

Ong, K. K., Chang, W. H., Tseng, Y. C., Lee, Y. S. and Lee, C. Y. (2002) 'A high throughput low cost context-based adaptive arithmetic codec for multiple standards'. *IEEE International Symposium on Image Processing* 1, 872-875

Opara, R. and Worgotter, F. (1996) 'Using visual latencies to improve image Segmentation'. *Neural Computation* 8, 1493–1520

Otair, M. A., Salameh , W. A. (2006) 'An Improved Back-Propagation Neural Networks using a Modified Non-linear function'. *The IASTED Conference on Artificial Intelligence and Applications*. Austria: Innsbruck

Ozkan, M., Dawant, B. M. and Maciunas, R. J. (1993) 'Neural network- based segmentation of multi-modal medical images—a comparative and prospective study'. *IEEE Trans. Medical Imaging* 12, 534–544

Paik, J. K. and Katsaggelos, A. K. (1993) 'Image restoration using a modi1ed Hopfield Network'. *IEEE Trans. On Image Processing* 1, 49–63

Pavlidis, G., Tsompanopoulos, A., Atsalakis, A., Papamarkos, N. and Chamzas, C. (2001) 'A Vector Quantization – Entropy Coder Image Compression System'. *IX Spanish Symposium on Pattern Recognition and Image Processing*

Pavlitov, K. and Mancler, O. (2004) 'FPGA Implementation of Artificial Neurons'. *Electronic Letters* 9, 22-24

Rabbani, M and Joshi, R (2002) 'An overview of the JPEG 2000 still image compression standard'. *Signal Processing: Image Communication* Elsevier Science B.V.17, 3-48

Rahman and Chowdhury, M. R. (2003) 'A New Approach for Compressing Color Images using Neural Network'. *Proceedings of International Conference on Computational Intelligence for Modeling, Control and Automation – CIMCA* Vienna: Austria

Razavi Behzad (2002) 'Design of Analog CMOS Integrated Circuits'. New Delhi: Tata McGraw-Hill

Robinson, J. and Kecman, V. (2003) 'Combining Support Vector Machine Learning With the Discrete Cosine Transform in Image Compression'. *IEEE Trans. on Neural Networks* 14 (4), 950-958

Roy, L. S. (1994) 'An Analog Neural Network with On-Chip Learning'. Unpublished M.Sc. Thesis. University of Oslo

Russo, L. E. and Real, C. E. (1992) 'Image Compression Using Outer Product Neural Network'. *Proc. IEEE International Conference on Acoustic Speech and Signal Processing* 2, 377-380

Rumelhart, D. E. and McClelland, J. L. (1986) 'Parallel Distributed Processing'. Cambridge: MIT Press

Ryan Kier, J., Reid Harrison, R. and Randall Beer, D. (2004) 'An MDAC Synapse for, Analog Neural Networks'. *IEEE Proc. On Circuits and Systems* 5 (1), 752-755

Sanger, T.D. (1989) 'Optimal unsupervised learning in a single layer linear feed forward neural network'. *IEEE Trans. On Neural Networks* 2, 459-473

Sarojini, S.M. (2001) *Image Compression using DWT-SPIHT* [Interview by P. Cyril Prasanna Raj] Trivandrum, Kerala, 10 March 2001

Schaphorst and Richard (1999) 'Videoconferencing and Videotelephony: Technology and Standards'. Norwood: Artech House Inc.

Shai, Cai-Qin, Geiger and Randy, L. (1987) 'A 5-v CMOS Analog Multiplier'. *IEEE Journal of solid state circuits* 22 (6), 1143-1146

Siaud, I., Morin B. (1999) 'Investigation on radio propagation channel measurements at 2.2 GHz and 3.5 GHz for the fixed wireless access in an urban area'. *Annual Telecommunication journal* 54 (9-10), 464-478

Sicuranzi, G. L., Ramponi, G. and Marsi, S. (1990) 'Artificial neural network for image Compression'. *Electronics Letters* 26, 477–479

Simon Haykin (2004) 'Neural Networks – A Comprehensive foundation'. Pearson Education

Signoroni, A., Lazzaroni, F. and Leonardi, R. (2003) 'Exploitation and extension of the region-of-interest coding functionalities in JPEG2000'. *IEEE Trans. on Consumer Electronics* 49 (4), 818– 823

Stuart Cheshire (1996) *It's the Latency, Stupid* [online] available from http://rescomp.stanford.edu/~cheshire/rants/Latency.html [24 Feb 2002]

Skrbek, M. (1999) 'Fast neural network implementation'. *Neural Network World* 9 (5), 375-391

Sonehara, N., Kawato, M., Miyake, S., Nakane, K. (1989) 'Image compression using a neural network model'. *International Joint Conf. on Neural Networks*, Washington DC

Steven, A.G. and Mario, J. G. (1993) 'Neural network implementation using reconfigurable architectures'. Selected papers from the Oxford 1993 international workshop on field programmable logic and applications on More FPGAs United Kingdom: Oxford. 443-451

Stuart, R. and Peter, N. (1980) 'Artificial Intelligence A Modern Approach'. Prentice Hall

Szu, H., Telfer, B. and Garcia, J. (1996) 'Wavelet transforms and neural networks for compression and recognition'. *IEEE Trans. on Neural Networks* 9 (4), 695-798

Szu, H., Telfer, B. and Kadambe, S. (1992) 'Neural network adaptive wavelets for signal

representation and classification'. Optical Engineering 31, 1907-1916

Thomas, M. C., Joy, A. T. (2006) 'Elements of Information Theory'. John Wiley & Sons

Thomos, N., Boulgouris, N. V., and Strintzis, M. G. (1992) 'Optimized Transmission of JPEG2000 Streams over Wireless Channel'. *IEEE Trans. on Image Processing* 15 (1), 54-67

Tiilikainen, M. P. (2001) 'A 14-bit 1.8-V 20-mW 1-mm2 CMOS DAC'. *IEEE Journal of Solid-State Circuit* 36, 1144-1147

Tomi Engdahl (2001) '*Telephone line audio interface circuits*'. [online]. Available from <http://www.epanorama.net/circuits/teleinterface.html> [20 July 2005]

Turner, C. and Peterson, L. (1992) 'Image Transfer: An End-to-End Design'. *Proceedings of ACM SIGCOMM Computer Communication* 22 (4), 258-269

Tzovaras, D. and Strintzis, M. G. (1998) 'Use of nonlinear principal component analysis and vector quantization for image coding'. *IEEE Trans. On Image Processing* 7, 1218– 1223

Veronin, C. P. and Priddy (1992) 'Optical image segmentation using neural based wavelet filtering techniques'. Optical Engineering 31, 287-294

Wallace, G. K. (1990) 'Overview of the JPEG (ISO/CCITT) still image compression Standard'. *Proceedings of the SPIE* 1244, 220–233

Wang, L. C., Rizvi, S. A. and Nasrabadi, N. M. (1998) 'A modular neural network vector predictor for predictive image coding'. *IEEE Trans. On Image Processing* 7, 1198– 1217

Wang, L., Fukatsu, Y. and Watanabe, K. (2001) 'A CMOS R-2R ladder digital-to-analog converter and its characterization'. *Proc. IMTC*, 1026-1031

Wang, J. S. and Wey, C. L. (1998) 'A 10-b, 100 MS/s, 2.8 mW CMOS switched-current DAC for low-power/low-voltage signal processing applications'. *IEEE Trans. On Circuit and Systems*, 526-529

Weingessel, A., Bischof, H. and Hornik, K. (1997) 'Adaptive combination of PCA and VQ networks'. *IEEE Trans. on Neural Networks* 8, 1208–1211

William, B. P., Joan, L. M. (2004) '*JPEG'*. Springer Publications

William Stallings (2007) '*Data and Computer Communications*', Prentice Hall

Wicker, S. B. (1995) 'Error Control Systems for Digital Communication and Storage', Upper Saddle River, NJ, Prentice Hall

Xiangjun, L. and Jianfei, C. (2007) 'Robust Transmission of JPEG2000 Encoded Images over Packet Loss Channels'. *ICME 2007*, 947-950

Zhang, L. (1996) 'Generating and coding of fractal graphs by neural network and mathematical morphology methods'. *IEEE Trans. On Neural Networks* 7 (2), 400-407

Appendix-A Backpropagation Algorithm

A.1 Introduction

Backpropagation is the generalization of the Widrow-Hoff learning rule to multiple-layer networks and nonlinear differentiable transfer functions. Input vectors and the corresponding target vectors are used to train a network until it can approximate a function, associate input vectors with specific output vectors, or classify input vectors in an appropriate way as defined by you. Networks with biases, a sigmoid layer, and a linear output layer are capable of approximating any function with a finite number of discontinuities. Standard backpropagation is a gradient descent algorithm, as is the Widrow-Hoff learning rule, in which the network weights are moved along the negative of the gradient of the performance function. The term backpropagation refers to the manner in which the gradient is computed for nonlinear multilayer networks.

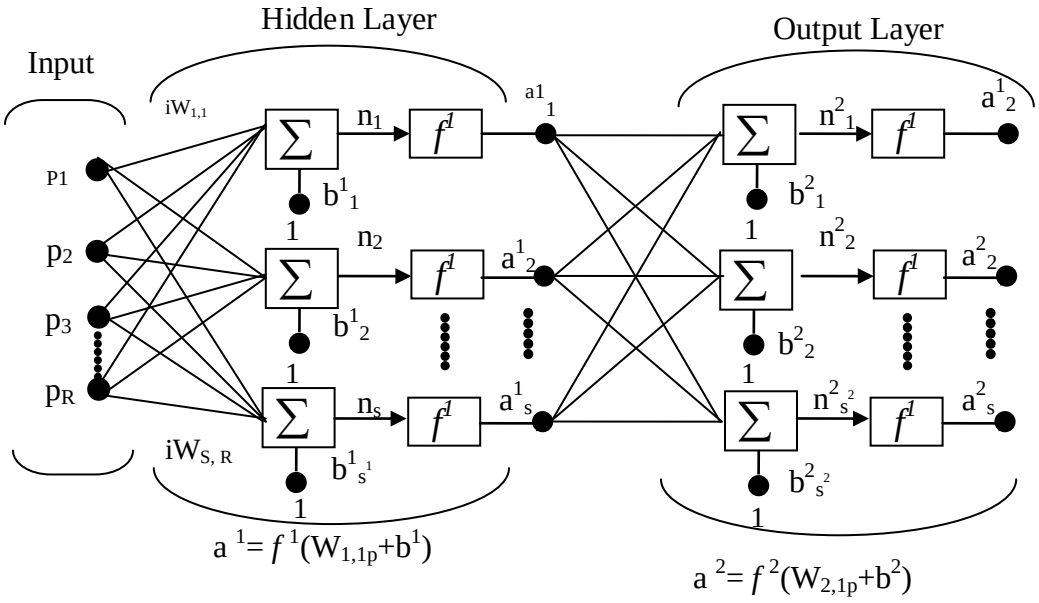


Figure A.1 Multilayer neural network architecture

There are a number of variations on the basic algorithm that are based on other standard optimization techniques, such as conjugate gradient and Newton methods. The discussion presented in this chapter is taken from Matlab Help and from Neural Network Design by

Hagan Demuth. To illustrate backpropagation training algorithm, the multilayered neural network (MLNN) shown in Fig. A.1 is considered, the MLNN consists of input layer, hidden layer and output layer.

Output of each layer is given by equation (A.1)

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1 \tag{A.1}$$

Where a is output of each layer, W is the weight matrix, bi is the bias, m represents network layer. Input layer has m = 0, output layer has m = M.

Network Training consists of following steps:

Step 1: Network Initialization

Inputs and targets are initialized along with initial weights and biases.

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Step 2: Forward Propagation

Input p is set as a^0, using the equation below, output of each layer is computed. The output of the final output layer is computed and is denoted as a. Equation A.2 presents these equations.

$$\mathbf{a}^{0} = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a} = \mathbf{a}^{M} \tag{A.2}$$

This is called as the forward propagation technique.

Step 3: Error calculation

The difference in network output a and expected target t is computed as in equation A.3, based on this error, the weight matrix gets updated.

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] \tag{A.3}$$

Step 4: Error sensitivity calculations and backpropagation

The sensitivities are computed by starting at the last layer, and then propagating backwards through the network to the first layer, as given in equation A.4.

$$\mathbf{s}^M \rightarrow \mathbf{s}^{M-1} \rightarrow \ldots \rightarrow \mathbf{s}^2 \rightarrow \mathbf{s}^1 \tag{A.4}$$

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{j=1}^{S^M} (t_j - a_j)^2}{\partial n_i^M} = -2(t_i - a_i)\frac{\partial a_i}{\partial n_i^M}$$

Above equations compute gradient and Jacobian matrix of the error matrix.

$$s_i^M = -2(t_i - a_i)\dot{f}^M(n_i^M)$$

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a}) \tag{A.5}$$

Sensitivity of each layer is computed as per equation A.4. Based on the computed sensitivity in the reverse direction i.e. from output to input layer, the weight matrix gets updated.

Step 5: Weight update

Based on the sensitivity factor, weight and bias values are update in every layer of the network as per equation A. 6.

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m(\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m \tag{A.6}$$

A.2 Software Reference Model for Network Training:

```
% image is read for compression
I1 = imread('C:\Program Files\MATLAB\R2007b\toolbox\images\imdemos\trees.tif');
I2 = imread('C:\Program Files\MATLAB\R2007b\toolbox\images\imdemos\pears.png');
I3=imread('C:\Program Files\MATLAB\R2007b\toolbox\images\imdemos\peppers.png');
I4 = imread('C:\Program Files\MATLAB\R2007b\toolbox\images\imdemos\trees.tif');
%size(I)
image(I1);
image(I2);
image(I3);
image(I4);
in1=I1(1:64,1:64);
in2=I2(1:64,1:64);
in3=I3(1:64,1:64);
in4=I4(1:64,1:64);
in1e=entropy(in1);
in2e=entropy(in2);
in3e=entropy(in3);
in4e=entropy(in4);
% image is displayed
% read part of the image for training
% in1=I1(65:128,129:192);
% in2=I2(65:128,129:192);
% in3=I3(1:64,1:64);
% in4=I4(1:64,1:64);
rr=64;
figure
r=4;
% part image is displayed
imshow(in1)
figure
imshow(in2)
figure
imshow(in3)
figure
imshow(in4)
% Image rearrangement
in1_rearrange=blkM2vc(in1,[r r]);
in2_rearrange=blkM2vc(in2,[r r]);
in3_rearrange=blkM2vc(in3,[r r]);
in4_rearrange=blkM2vc(in4,[r r]);
```

```
in_combined = [in1_rearrange in2_rearrange in3_rearrange in4_rearrange in1_rearrange
in2_rearrange in3_rearrange in4_rearrange];
%in_combined = [in1_rearrange in1_rearrange in1_rearrange in1_rearrange
in1_rearrange in1_rearrange in1_rearrange in1_rearrange];
% normalizing the input to less than and equal to 1
in_combined_normalised=in_combined/255;
% input is set as the target
target=in_combined_normalised;
% creating a neural network having 4 input layer and 16 output layer,
% tansig is input layer transfer function
% purelin is output layer transfer function
% to train the network use train rp function which is backpropagation
% technique
net_c=newcf(minmax(in_combined_normalised),[8 7 8
16],{'tansig','tansig','purelin','purelin'},'trainrp');
% training constraints
net.trainparam.show=5;
net.trainparam.epochs=100;
net.trainparam.goal=1e-5;
% train the network
[net_s,tr]=train(net_c,in_combined_normalised,target);
% first image for testing
image1test= in1_rearrange/255;
a=sim(net_s,image1test);
% rearrange the matrix for display
a4=vc2blkM(a,r,64);
% scale the output to original size
asc=a4*255;
% convert to unsigned number
az=uint8(asc);
% display the output
figure
imshow(az);

%second image for testing
image2test= in2_rearrange/255;
a2=sim(net_s,image2test);
% rearrange the matrix for display
a42=vc2blkM(a2,r,64);
% scale the output to original size
asc2=a42*255;
% convert to unsigned number
```

```
az2=uint8(asc2);
% display the output
figure
imshow(az2);

%third image for testing
image3test= in3_rearrange/255;
a3=sim(net_s,image3test);
% rearrange the matrix for display
a43=vc2blkM(a3,r,64);
% scale the output to original size
asc3=a43*255;
% convert to unsigned number
az3=uint8(asc3);
% display the output
figure
imshow(az3);

%fourth image for testing
image4test= in4_rearrange/255;
a4=sim(net_s,image4test);
% rearrange the matrix for display
a44=vc2blkM(a4,r,64);
% scale the output to original size
asc4=a44*255;
% convert to unsigned number
az4=uint8(asc4);
% display the output
figure
imshow(az4);

%first image
imageMaxError1 = max(max(abs(double(in1)-double(az))))
imageMSE1 = sum(sum((double(in1)-double(az)) .^ 2)) / size(in1,1) / size(in1,2)
psnr1 = 10*log10(255*255/imageMSE1)

%second image
imageMaxError2 = max(max(abs(double(in2)-double(az2))))
imageMSE2 = sum(sum((double(in2)-double(az2)) .^ 2)) / size(in2,1) / size(in2,2)
psnr2 = 10*log10(255*255/imageMSE2)

%third image
```

```
imageMaxError3 = max(max(abs(double(in3)-double(az3))))
imageMSE3 = sum(sum((double(in3)-double(az3)) .^ 2)) / size(in3,1) / size(in3,2)
psnr3 = 10*log10(255*255/imageMSE3)

%fourth image
imageMaxError4 = max(max(abs(double(in4)-double(az4))))
imageMSE4 = sum(sum((double(in4)-double(az4)) .^ 2)) / size(in4,1) / size(in4,2)
psnr4 = 10*log10(255*255/imageMSE4)
```

Appendix-B

B.1 Digital implementation of neural network architecture

In this section, discussion on FPGA implementation of neural network architecture for image compression and decompression is presented. Input image of size 128 x 128 is sub divided into 8 x 8 blocks of sub images. Each 8 x 8 block is rearranged to 64 x 1 inputs. The neural network architecture selected for digital implementation is hybrid architecture, the hidden layer has tansig as the network function and the output layer has purelin as the network function. The hidden layer consists of 4 neurons with 4 bias elements and the output layer consists of 64 neurons with 64 bias elements. 64 x 1 input is compressed to 4 x 1 at the hidden layer, and the 4 x 1 output is decompressed to 64 x 1 at the output layer. The hidden layer and the output layer consist of 256 weights each that have been obtained after training. Fig. B.1 shows the neural network architecture that is implemented using FPGA.

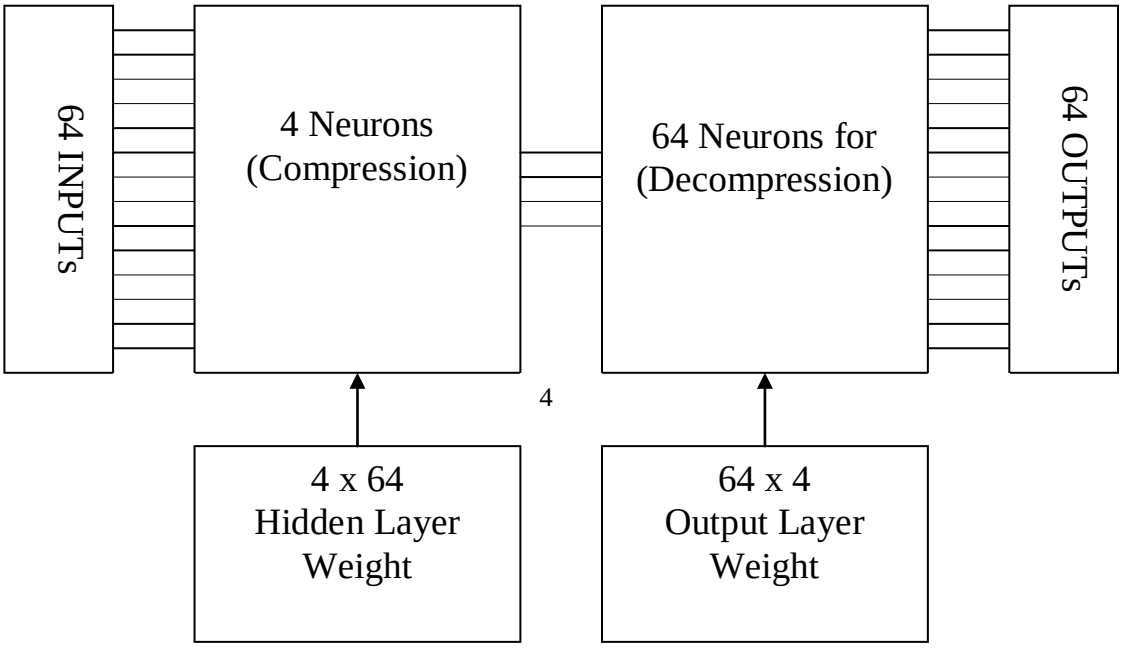


Figure B.1 NN architecture for image compression

To generalize the network for compression and decompression of any image, 20 sets of image samples are considered having all the properties of an image such as vertical lines, horizontal lines, diagonals, curves, edges, plain surfaces and sharp edges. The network is trained using these data sets, so that the weight matrix obtained can be used to compress and decompress with any unknown image. FPGA implementation of neural network architecture shown in Fig. B.2, it is requried to identify the building blocks of the architecture. The hidden layer consists of 4 neurons, with each neuron computing one output (z) as in equation B.1,

$$n_1 = x_1w_{1,1} + x_2w_{2,1} + x_3w_{3,1} + \ldots\ldots\ldots\ldots\ldots\ldots x_{64}w_{64,1} + b_{1h} \quad \text{(B.1)}$$

$$z_1 = f(n_1) = \text{tansig}(n_1)$$

x is the input pixel, w is the weight, b_{1h} is the bias in the hidden layer, n is the intermediate output and z is the final output of hidden layer neuron.

The output layer consists of 64 neurons, with each neuron computing one output as in equation B.2,

$$m_1 = z_1w_{1,1} + z_2w_{2,1} + z_3w_{3,1} + z_4w_{4,1} + b_{1o} \quad \text{(B.2)}$$

$$x_1 = f(m_1) = \text{purelin}(m_1) = m_1$$

b_{1o} is the bias in the output layer, m is the intermediate output, x is the output at the output layer of the neuron.

Fig. B.2 shows the architecture of single neuron in the hidden layer and Fig. B.3 shows the architecture of two neurons in the output layer.

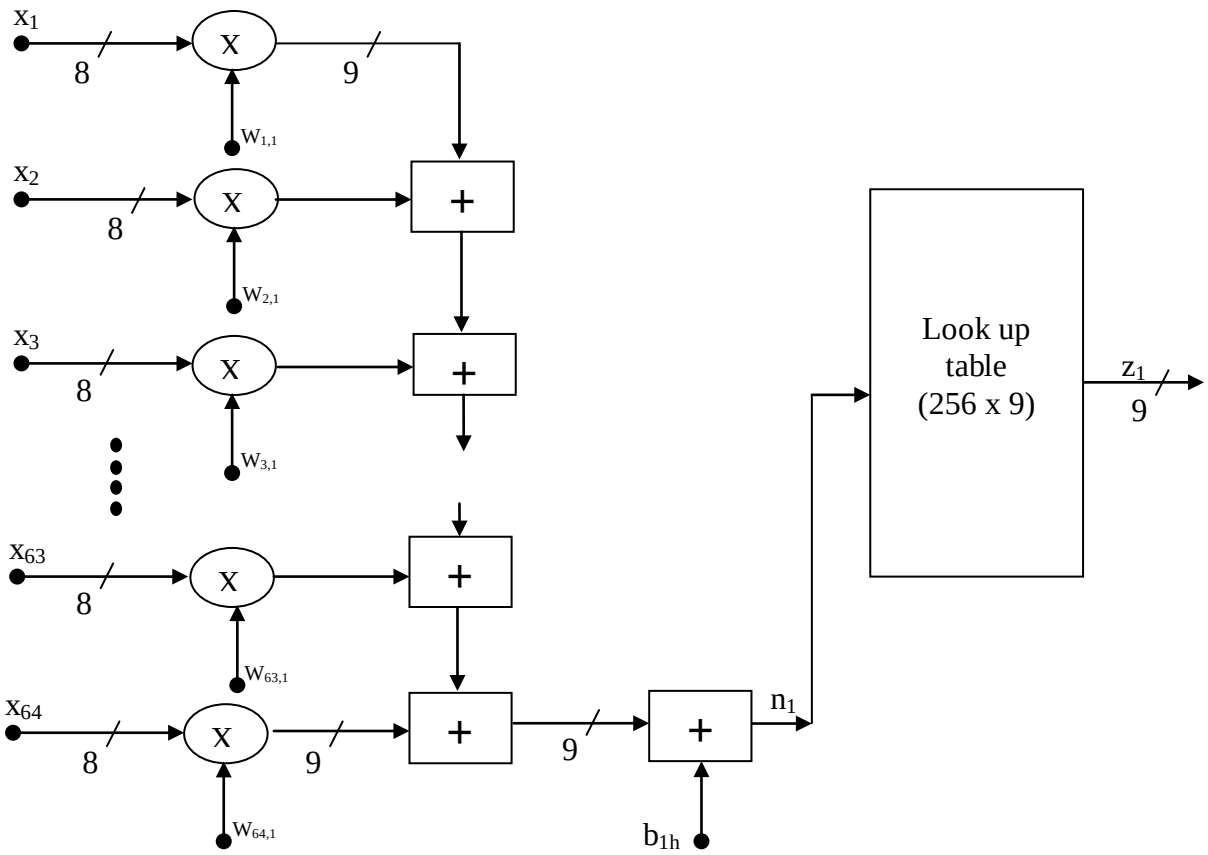


Figure B.2 Single neuron architecture of hidden layer

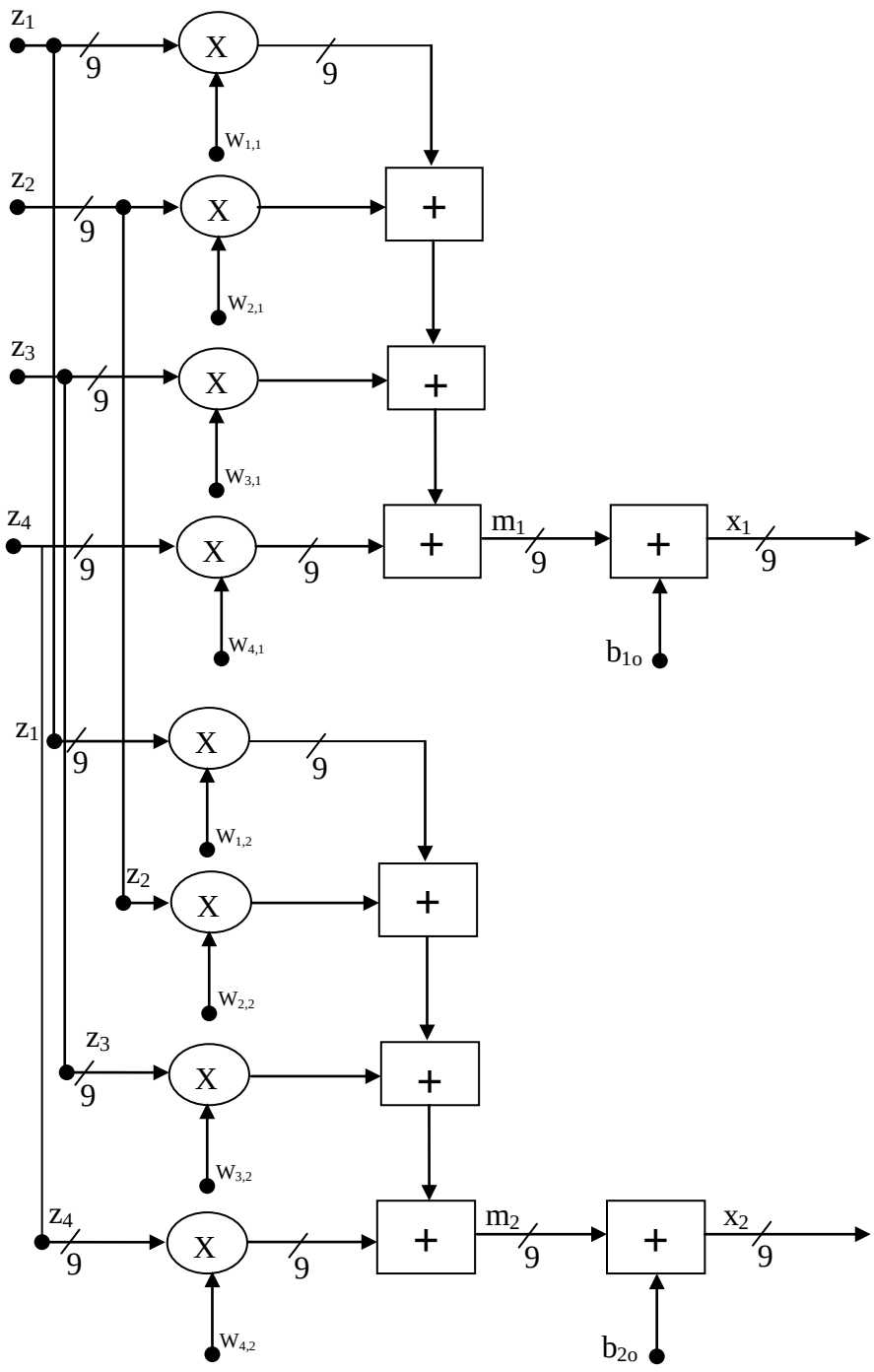


Figure B.3 Output Layer neural network architecture (2 neurons)

Based on the discussion presented in Appendix-A, the network is trained with various images as shown in Fig. B.4. Various possible images that consists of vertical lines, horizontal lines, curves, diagonal lines and circles have been used for training. Also general images that have been adopted for validating image processing algorithms have also been used for training the network.

These images have been removed

Figure B.4 Images for training and testing neural network architecture

The weights and biases for the neural network architecture are obtained after training in Matlab and are presented in Table 5.1. The weights and biases are scaled to integer values and represented using 2's complement number system and stored in memory. For FPGA implementation, the input operand size is 8-bit, weights and biases are represented using 8-bit 2's complement number representation. The hidden layer consists of network function that is realized using tansig. In this work, tansig function is realized using look up table approach. The outputs of tansig function for various values of input in the range +/- 127 is computed and stored in ROM. The sum of product output of each neuron is used to access corresponding memory content of ROM. The sum of product output is used as address to the ROM. The output layer does not require look up table as the network function is a linear function. Due to scaling and rounding of weights and biases using 8-bit number representation, the quantization leads to a maximum loss in weight values of $+/-(1/2^7)$. From the simulation results obtained using ModelSim, it is observed that due to scaling of weights, biases in the hidden layer and output layer, there is a maximum difference of +/- 10. This is observed based on the Matlab simulation results and ModelSim simulation results considering individual pixel values. Randomly chosen pixels values from Matlab and ModelSim results have been compared to find the mismatches between software and FPGA implementation. The image quality measured in terms of MSE and PSNR are also compared Fig. B.5 shows the results of Matlab and ModelSim simulation. For functional verification of HDL model, HDL co-simulation is carried out by interfacing Matlab with ModelSim.

This image has been removed

Figure B.5 Matlab and ModelSim results of decompressed image

The architecture complexity of input layer, hidden layer and output layer are presented in Table B.1. The architecture shown in Fig. B.2 and Fig B.3 requires sub blocks such as multipliers, adders, look-up table, registers and control unit for realization. Table B.1 presents the complexity of neural network architecture in terms of number of sub blocks required for hardware realization.

Table B.1 Complexity of neural network architecture

	Number of neurons	Number of multiplier	Number of adders	Number of look up table	Number of registers in the input layer (x+ w+ b+ z)	Number of registers in the output layer (z+ w+ b+ x)
Hidden layer	4	256	256	4	64 + 64 +4 + 4 = 136	------
Output layer	64	256	256	0	--------	4 + 64 + 64 + 64 = 196

Multipliers and adders are the major building blocks of neural network architecture, in this work various multipliers and adders are modeled using Verilog and synthesized using Xilinx ISE targeting Spartan IIIE FPGA. In general multiplication involves two basic operations:

1. Generation of partial products
2. Accumulation of partial products

Performance and speed depends on how we generate partial products, techniques we use to reduce the partial products and accumulate them. Based on these points a number of researches are in progress. Multipliers can be basically classified into two three categories

1. Sequential multiplier – This involves generating partial products sequentially and adds each newly generated product to previously accumulated partial product
2. Parallel multiplier – It generates partial products in parallel, accumulates using a fast multi-operand adder
3. Array multiplier - array of identical cells generating new partial products; accumulating them simultaneously
    - No separate circuits for generation and accumulation
    - Reduced execution time but increased hardware complexity

FPGA implementation involving translation, mapping, placement & routing as well as device configuration are important steps in FPGA implementation. Implementation of multiplier should deal with all the constraints like area, timing and power. The XST tools of Xilinx automatically tries to optimize the design by removing unconnected or non operational blocks, even though timing constraints cannot be optimized well using Xilinx tools it provides the basic information of timing. The summary of implementation results is presented in Table B.2.

Table B.2 Comparison of multiplier architectures

Multiplier type	Number of Slices utilized out of 1920	Delay in ns
BCSD	156	11.76
Array	143	39.27
Modified Booths	91	24.21
Baugh Wooley	144	44.12
Wallace tree	186	18.46

From the results obtained and presented in Table B.1, BCSD and Wallace Tree multipliers are selected for FPGA implementation as they are faster than other multipliers. In case of BCSD multiplier, the input, output, weight and bias elements need to be represented using BCSD number format, thus the entire neural network architecture

need to be represented using BCSD number system. This requires additional logic at the input and output of neural network architecture to encode and decode inputs and outputs respectively from 2's complement to BCSD number format. For simplicity, Wallace tree multiplier is used to realize the neural network architecture. There are various adder circuits reported in literature for FPGA implementation, in this work Xilinx adder logic that uses carry chain logic is adopted for architecture implementation.

A single neuron in the hidden layer is modeled using Verilog HDL and is synthesized using Xilinx ISE tool. Based on the synthesize results obtained it is found that the selected FPGA cannot support implementation of 64-4-64 neural network architecture. Hence, Virtex-5 device from Xilinx has been chosen for FPGA implementation. HDL modeling for the 64-4-64 neural network architecture is modeled and synthesized using Xilinx ISE targeting Virtex-5 FPGA. The synthesized results are summarized in Table B.3.

Table B.3 Synthesis results of 64-4-64 neural network architecture

Virtex-5 FPGA (XC5VLX110FF676)	Number of slices out of 51840	Delay in ns	Power consumption in W
64-4-64 network	4132	4.406	0.013

Further ASIC implementation of single neuron is carried out targeting 130nm TSMC technology file. HDL model for single neuron cell is synthesized using Design Compiler and timing analysis is carried out using Prime Time. FPGA and ASIC synthesis results of single neuron cell are presented in Table B.4.

Table B.4 Single neuron comparison

Parameter	FPGA implementation(Spartan IIIE)	ASIC results
Gate count Number of 2 input NAND gates	25520	1556
Maximum delay in ns	56	4
Dynamic power dissipation in mW	35	2

ASIC implementation of 64-4-64 network architecture can be further attempted optimizing area, power and speed performances. Low power techniques and area optimization techniques can be attempted targeting 65nm CMOS technology. One of the major observations made during functional verification of neural network architecture for image compression and decompression is that, input image being sub-divided into 8x8 sub-blocks and is compressed and reconstructed at the output layer to its original size checker box errors occur in the reconstructed image as shown in Fig. B.6.

Figure B.6 Results of image reconstruction using neural network architecture

In order to overcome checker box errors, Wilford Gillespie proposed still image compression technique using Discrete Wavelet Transform (DWT) and Neural Networks. In this architecture, images are decomposed into sub-band components using DWT, based on the sub bands obtained image is grouped into multiple sub-bands and is used in training the network. Due to decomposition of images into various sub-bands of frequency components, and neural network compressing the regrouped sub band components, checker block errors are minimized in the reconstructed image. The author

compared the MSE of the results obtained that with JPEG based results, and was able achieve very less improvement in MSE (MSE reduced from 2.9643 to 2.9586). This is not a very significant improvement; hence to improve the MSE of the DWT-NN architecture a modified DWT-NN architecture called as hybrid architecture is proposed and implemented. Fig. B.7 shows the hybrid architecture for image compression and decompression using DWT and NN.

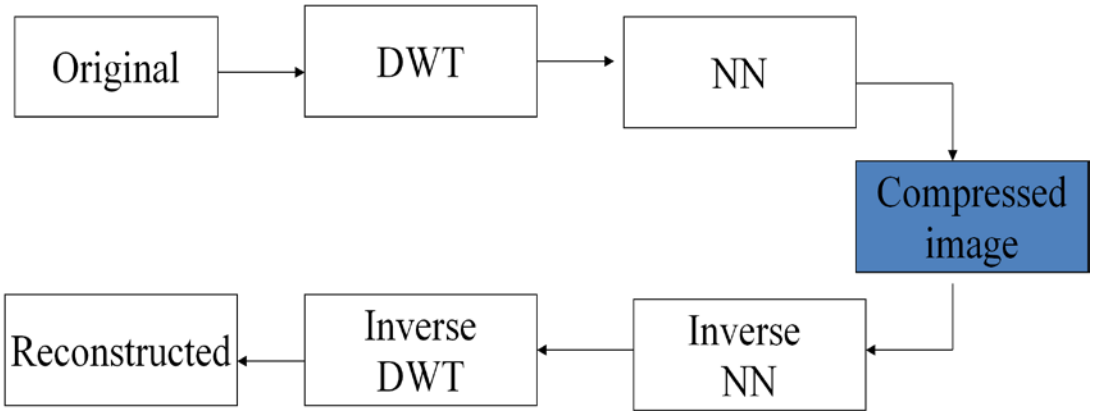


Figure B.7 Hybrid architecture for image compression and decompression

In the modified architecture, the decomposed image after DWT is ordered such that the sub band frequency components are arranged such that the corresponding frequency sub-bands are placed adjacent prior to compression using neural networks. The reordered sub-bands are used in raining the network, thus optimum weights and biases are obtained for compression and decompression. This modified reordering of sub-band components has improved the performance of the hybrid architecture. Fig. B.8 shows the modified reordering scheme. Fig. B.9 presents the Matlab simulation results of hybrid architecture. In the reordered scheme, the numbers indicate the arrangement matrix of sub-band components.

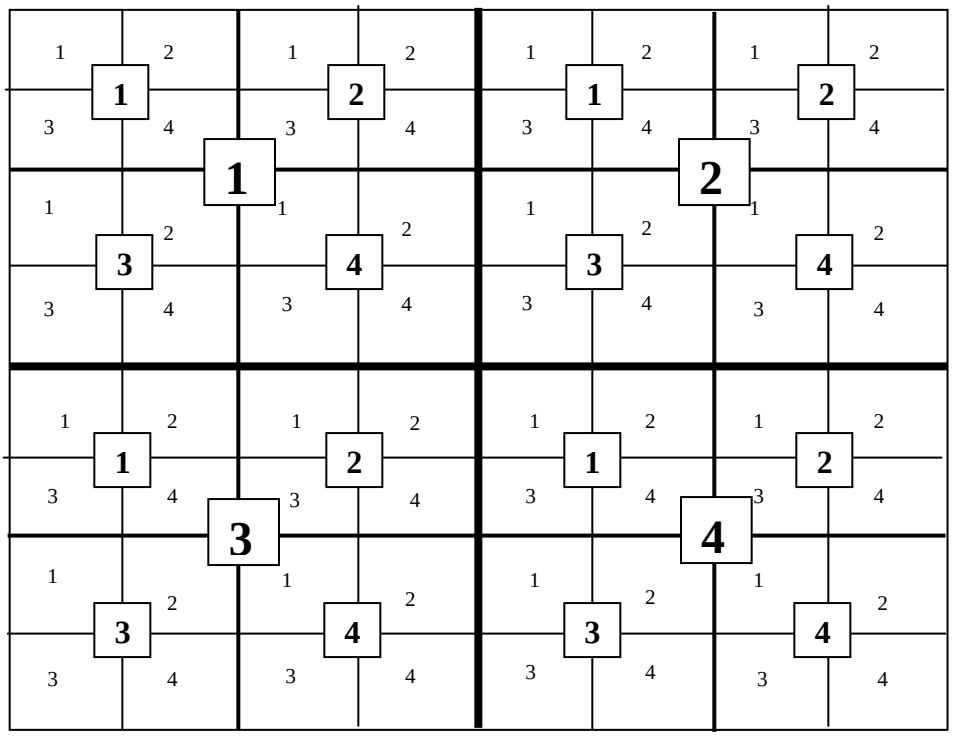


Figure B.8 Modified reordering scheme of sub-bands for NN training

These images have been removed

Figure B.9 Simulation results of hybrid architecture

The results obtained can be further improved by choosing appropriate wavelets and training the network for various test images. FPGA implementation and ASIC implementation of hybrid architecture is carried out, the results obtained have been published in IEEE conference on Biomedical Signal Processing held in Singapore during December 2008.

Appendix – C Entropy

In information theory, entropy is a measure of the uncertainty associated with a random variable. The Shannon entropy is a measure of the average information content one is missing when one does not know the value of the random variable. The concept was introduced by Claude E. Shannon (1948) in his paper "A Mathematical Theory of Communication". Based on the discussion provided by Shannon, the entropy H of a discrete random variable X with possible values $\{x_1, ..., x_n\}$ is given by:

$$H(X) = E(I(X)) \quad \text{(C.1)}$$

Where, E is the expected value function, and $I(X)$ is the information content or self-information of X. $I(X)$ is random variable and p denotes the probability mass function of X then the entropy can explicitly be given by:

$$H(X) = \sum_{i=1}^{n} p(x_i) I(x_i) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i) \quad \text{(C.2)}$$

Where, b is the base of the logarithm used. For digital applications, b is 2. The ***b*-ary entropy** of a source $S = \{a_1, ..., a_n\}$ and discrete probability distribution $P = \{p_1, ..., p_n\}$ where p_i is the probability of a_i is defined by:

$$H_b(S) = -\sum_{i=1}^{n} p_i \log_b p_i \quad \text{(C.3)}$$

The discussion presented is taken from wikipedia.org.

In Matlab, there are inbuilt functions provided that can assist in calculating entropy of a given matrix. The syntax for computing Entropy of a given input is

E = entropy(I)

Where, E is the entropy, I is the input matrix. For a given input I, entropy(I) returns a scalar value representing the entropy of gray scale image I. For a given image Entropy is a statistical measure of randomness that can be used to characterize the texture of the input image. To compute entropy using Matlab, histogram of the image is required. Entropy is defined as:

E = -sum (p.*$\log_2$(p)), where p contains the histogram for a given image.

Appendix – D Spice Code Model for Analog Neural Network

Gilbert cell multiplier

```
.subckt mult v11 v1g v21 v2g 1 2 vb1 vdd vee
m1 1 vb1 vdd vdd modp l=1u w=120.7u
m2 2 vb1 vdd vdd modp l=1u w=120.7u
m3 1 v11 3 3 modn l=1u w=4u
m4 2 v1g 3 3 modn l=1u w=4u
m5 1 v1g 4 4 modn l=1u w=4u
m6 2 v11 4 4 modn l=1u w=4u
m7 3 v21 5 5 modn l=1u w=2u
m8 4 v2g 5 5 modn l=1u w=2u
m9 5 vb1 vee vee modn l=1u w=44.5u
m10 1 1 vdd vdd modp l=1u w=120u
m11 2 2 vdd vdd modp l=1u w=120u
.ends
```

Differential amplifier

```
.subckt tan v1 v2 1 2 vb1 vdd vee
m1 1 vb1 vdd vdd modp l=1u w=1.5u
m2 2 vb1 vdd vdd modp l=1u w=1.5u
m3 1 v2 3 3 modn l=1u w=2u
m4 2 v1 3 3 modn l=1u w=2u
m5 3 vb1 vee vee modn l=1u w=3.3u
.ends
```

Neuron Activation function

```
.subckt fun v1 v2 na1 na0 op1 vb1 vdd vee vsb
m1 1 vb1 vdd vdd modp l=1u w=16u
m2 2 vb1 vdd vdd modp l=1u w=16u
m3 3 v1 1 1 modp l=1u w=2u
m4 na1 v1 2 vsb modp l=1u w=4u
m5 na0 v2 1 vsb modp l=1u w=4u
m6 4 v2 2 2 modp l=1u w=2u
m7 3 3 vee vee modn l=1u w=4u
m8 na1 3 vee vee modn l=1u w=4u
m9 na0 3 vee vee modn l=1u w=4u
m10 4 4 op1 op1 modn l=1u w=4u
m11 op1 op1 vr4 vr4 modn l=1u w=2u
m12 vr4 vr4 vee vee modn l=1u w=2u
.ends
```

Neural Architecture

```
*neurons in first layer
x1 v11 v10 w111 w110 1 2 vb vdd vee mult
x2 v21 v20 w121 w120 1 2 vb vdd vee mult
x3 1 2 o11 o10 d1 vb1 vdd vee vsb fun
x4 v11 v10 w131 w130 3 4 vb vdd vee mult
x5 v21 v20 w141 w140 3 4 vb vdd vee mult
```

x6 3 4 o21 o20 d2 vb1 vdd vee vsb fun
x7 v11 v10 w151 w150 5 6 vb vdd vee mult
x8 v21 v20 w161 w160 5 6 vb vdd vee mult
x9 5 6 o31 o30 d3 vb1 vdd vee vsb fun
***output layer**
x10 o11 o10 w211 w210 7 8 vb vdd vee mult
x11 o21 o20 w221 w220 7 8 vb vdd vee mult
x12 o31 o30 w231 w230 7 8 vb vdd vee mult
x13 7 8 op1 op0 d4 vb1 vdd vee vsb fun
